RANDOMIZED ALGORITHMS IN SOME COMMUTATIVE AND NONCOMMUTATIVE DOMAINS

by

Pushkar S. Joglekar

THE INSTITUTE OF MATHEMATICAL SCIENCES, CHENNAI.

A thesis submitted to the
Board of Studies in Mathematical Sciences

In partial fulfillment of the requirements

For the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

September 2011

Homi Bhabha National Institute

Recommendations of the Viva Voce Board

As members of the Viva Voce Board, we recommend that the dissertation prepared by **Pushkar S. Joglekar** entitled "Randomized Algorithms in some Commutative and Noncommutative Domains" may be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

---------- **Date :**
Chairman and Convener of Committee: V. Arvind

---------- **Date :**
Member : Somenath Biswas

---------- **Date :**
Member : Meena Mahajan

---------- **Date :**
Member : Venkatesh Raman

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to HBNI.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it may be accepted as fulfilling the dissertation requirement.

---------- **Date :**
Guide : V. Arvind

DECLARATION

I hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and the work has not been submitted earlier as a whole or in part for a degree/diploma at this or any other Institution or University.

Pushkar S. Joglekar

ACKNOWLEDGEMENTS

I take this opportunity to express my gratitude towards various people who influenced my academic and personal life over the years.

First of all, I am most grateful to my advisor V. Arvind. I would like to thank him for his kind support and the guidance which led to this thesis. The courses he taught like Algebra and Computation, Computational Complexity led me to pursue the research in this area. Whenever I had been stuck with some mathematical or even a personal problem, a discussion with Arvind used to boost up my confidence and used to make me think positively, I can not thank him enough for that. He not only helped me to improve my knowledge about the computer science but also he spent his precious time to help me to improve my technical writing ability and the presentation skills. His clarity of exposition, his ability to connect seemingly different problems in the complexity theory and the deep insights into theoretical computer science will always remain inspirational for me throughout my research career.

I would like to thank all the faculty members in the TCS group at IMSc for teaching various wonderful courses. Particularly my sincere thanks to V. Arvind, Meena Mahajan, Kamal Lodaya, R. Ramanujam, Venkatesh Raman, C. R. Subramanian. I would like to thank Meena for organizing interesting theory seminars and teaching interesting courses like Computational Complexity and Computational Geometry. My sincere thanks to Jam and Kamal for introducing me to interesting areas of Logic, Game Theory and Automata Theory. I thank Venkatesh and CRS for the interesting courses on Randomized Algorithms, Combinatorics. I take this opportunity to thank R. Balasubramanian, the director of IMSc for providing excellent academic environment. The administrative staff at IMSc was very helpful, I thank them for their co-operation.

I have been introduced to interesting areas of Mathematics by my teacher Ram Darade when I was at the Junior College which motivated me to pursue research in Theoretical Computer Science in the later years, my sincere thanks to him for his great teaching.

Though I did not have many friends during the years I spent at IMSc, those I had, I shared many nice moments with them. Raghu, Aravind, Partha, Prajakta, Bireswar, Somnath, Sarbeswar, Samrat, Jayalal, Maruthi, Gaurav made my stay at the IMSc a

memorable one. I thank them for the fun moments shared at the IMSc. Special thanks to Srikanth for collaborating with me on nice research problems. I thank Aravind, Raghu, Partha, Jayalal for several interesting discussions.

I thank my family members, aai, baba, aaji, aajoba, and kedar for their kind support. My father nourished my interest in Mathematics starting from my school days by teaching me various nice results in Mathematics and providing lots of interesting books on Mathematics. He will always remain the constant source of motivation for me. I thank my mother for her love and the things she taught me since my childhood. Kedar, my younger brother and the best friend, gave me the motivation at several crucial times. I can not thank him enough for his support, for the wonderful discussions we had on several topics ranging from Philosophy to Mathematics. This work was not possible without his encouragement.

Finally, I thank aaji and aajoba (my grand parents) for their love and the teachings. This work is dedicated to my grandmother Smt. Sushila Govind Joglekar and my grandfather Late Shri. Govind Anant Joglekar.

Abstract

In this thesis we explore the computation complexity of some algebraic problems in the commutative and the noncommutative setting. Our motivation is to better understand the algorithmic questions in both the settings and to see the interplay between them. We also investigate the possibility of applying the techniques and tools developed in the one model to the other. Specifically, we focus on the computational complexity of the problems over integer lattices, permutation groups and arithmetic circuits.

Algorithmic problems over integer lattices and permutation groups Shortest vector problem(SVP) and the closest vector problem(CVP) are two important problems over integer lattices and their algorithmic complexity is a subject of extensive research in the recent time due to advent of lattice based cryptosystems. Both of these problems are known to be NP-hard. Ajtai, Kumar and Sivakumar in a breakthrough work gave a singly exponetial time randomized algorithm for SVP and a singly exponential algorithm for solving CVP within factor of $1 + \epsilon$ for any constant $\epsilon > 0$. Recently a new problem was introduced by Blömer and Naewe called *Subspace avoiding problem* SAP to better understand the computational complexity of CVP and SVP. Both of these problems are special cases of SAP. Given an integer lattice $\mathcal{L}$ of rank n and a subspace $M \subset \mathbb{R}^n$ of dimension k, the Subspace avoiding problem is to compute the length of a shortest vector in $\mathcal{L} \setminus M$ with respect to the concerned norm. In this thesis we give a new algorithm for SAP based on the Ajtai-Kumar-Sivakumar sieving technique which performs better compared to Blömer and Naewe algorithm parameterized on the dimension k of the subspace concerned. Our algorithm works for metrics given by gauge functions which includes usual ℓ_p norms. Later we give some applications of our algorithm to the CVP and the SVP problem.

Next we investigate the computational complexity of two natural problems for metrics on permutation groups (which are nonabelian in general) given by generating sets. These problems are exact analogue of closest vector problem and the shortest vector problem. These problems are also known to be NP-hard for various metrics. Interestingly we can adapt Ajtai-Kumar-Sivakumar like sieving technique to give a singly exponential algorithm to compute a shortest non-

identity permutation in a given permutation group with respect to ℓ_∞ metric. We also extend some of the results known for CVP and SVP to the permutation group setting, some of our results need a restriction on the group to be solvable(which are also nonabelian in general).

Monomial algebras and finite automata In this part of the thesis we study arithmetic circuit and algebraic branching program size lower bound questions as well as polynomial identity testing problem over *monomial algebras* both in the noncommutative and the commutative setting. Main tool we use is basic automata theory. Our first result is extension of Nisan's lower bound for the Permanent and Determinant polynomials over free noncommutative algebra to the similar lower bound result over noncommutative monomial algebras. Furthermore, the Raz-Shpilka deterministic identity test for noncommutative ABPs also carry over to monomial algebras.

In the commutative setting, we extend Jerrum and Snir's $2^{\Omega(n)}$ size lower bound for monotone arithmetic circuits computing the $n \times n$ Permanent in the commutative polynomial ring to similar lower bound result over commutative monomial algebras. Next we investigate randomize parallel complexity of Monomial Search Problem which is a natural search version on the identity testing problem. We give randomized-NC^2 upperbound on the complexity both in the commutative and noncommutative setting.

Hadamard product of polynomials We introduce and study the Hadamard product of the multivariate polynomials in the free noncommutative polynomial ring $\mathbb{F}\{x_1, x_2, \cdots, x_n\}$. We explore arithmetic circuit and branching program complexity of the Hadamard product of polynomials when they are individually given by arithmetic circuits and/or algebraic branching programs. We show that the *noncommutative* branching program complexity of the Hadamard product of polynomials given by ABPs is upper bounded by the product of the given branching program sizes. We then apply this result to tightly classify the complexity of identity testing problem for noncommutative ABPs over field of rationals. We show that the problem is complete for logspace counting class $\mathrm{C_{=}L}$. We also explore same problem over finite fields and show nonuniform-$\mathrm{Mod}_p\mathrm{L}$ upperbound on the complexity.

Contents

List of Figures

1

Introduction

In this thesis we explore the algorithmic complexity of some algebraic problems in both the commutative and the noncommutative settings. Our motivation is to better understand algorithmic questions in both the settings and to see the interplay between them.

Commutative and noncommutative computation exhibit an interesting difference in computational complexity. For example, in the commutative setting computing the determinant has very efficient parallel algorithms (e.g. [MV97]). These algorithms actually describe polynomial-size algebraic branching programs for computing the determinant polynomial. On the other hand, in the noncommutative setting, Nisan [N91] has shown exponential size lower bounds on the size of any algebraic branching program which computes the determinant. In fact, more recently [AS10] it is shown that it is unlikely that the noncommutative determinant has a polynomial-sized arithmetic circuit; indeed, the existence of such a circuit would imply that the noncommutative permanent polynomial has a polynomial-sized arithmetic circuit, which in turn gives a polynomial-sized arithmetic circuit for the commutative permanent polynomial, which is widely believed to be false.

To see another example of the contrast between these two models consider the polynomial identity testing problem for algebraic branching programs and arithmetic circuits. For algebraic branching programs, this problem has a deterministic polynomial time algorithm [RS05] in the noncommutative setting, whereas in the commutative setting,

getting such an algorithm is a long standing open problem. In the case of arithmetic circuits, there is a randomized polynomial time algorithm based on Schwartz-Zippel lemma for the problem in the commutative setting whereas in the noncommutative setting, the problem can be solved in randomized polynomial time only if the polynomial computed by the given arithmetic circuit has polynomial (in the number of input variables) degree.

In this thesis we pursue this direction of research further and compare the complexities of various algebraic problems in the commutative and the noncommutative domains. We also investigate the possibility of applying the techniques and tools developed in the one model to the other. Specifically, we focus on the computational complexity of the problems over integer lattices, permutation groups and arithmetic circuits. Now we describe the main results in this thesis.

1.1 Sieving Algorithms for Lattice Problems

Lattices are geometric objects that can be pictorially described as the set of intersection points of an infinite regular grid in n dimensions. More precisely, given linearly independent vectors $b_1, \ldots, b_n \in \mathbb{R}^n$ the lattice $\mathcal{L}$ generated by them is the set of all integer linear combinations of b_i's i.e. $\mathcal{L} = \{\Sigma_{i=1}^{n} \alpha_i b_i | \alpha_i' s \in \mathbb{Z}\}$. Despite their apparent simplicity, lattices have a rich combinatorial structure which leads to numerous applications in mathematics and computer science.

Two fundamental algorithmic problems concerning integer lattices are the *shortest vector problem* (SVP) and the *closest vector problem* (CVP). Given a lattice $\mathcal{L} \subset \mathbb{R}^n$ by a basis, the shortest vector problem (SVP) is to find a shortest non-zero vector in $\mathcal{L}$ with respect to a given metric. Likewise, the closest vector problem (CVP) takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a vector $v \in \mathbb{R}^n$ and asks for a $u \in \mathcal{L}$ closest to v with respect to a given metric.

The study of lattices from the computational point of view was marked by a major breakthrough: The LLL algorithm developed by Lenstra, Lenstra and Lovasz [LLL82] which gives an approximate solution for SVP in n dimensions. Given a rank n integer lattice, in deterministic polynomial time the LLL algorithm outputs a nonzero vector

$v \in \mathcal{L}$ whose ℓ_2 norm is guaranteed to be within a $2^{O(n)}$ factor of the norm of a shortest nonzero vector in $\mathcal{L}$. The LLL algorithm also can be used to solve CVP within a $2^{O(n)}$ factor [Bab86].

Despite the relatively poor quality of the approximate solution in the worst case, the LLL algorithm allows us to devise polynomial time algorithms for various problems including polynomial factorization over rationals, breaking a knapsack-based cryptosystem, solving integer linear programs in a fixed number of variables etc [LLL82, MG02]. There are some results which improve the approximation factor in the LLL solution to a slightly sub-exponential factors (e.g. [Sch94]).

NP-hardness of CVP (in any ℓ_p norm) and SVP (in ℓ_∞ norm) was originally proved by van Emde Boas in 80's [Bos81]. In fact, it is known that, even finding an approximate solution for CVP is a hard problem (e.g. [ABSS97], [DKS98]). In [DKS98] it is shown that CVP is NP-hard to approximate within approximation factor of $2^{O(\frac{\log n}{\log \log n})}$. Showing NP-hardness for CVP for polynomial approximation factor is an important open problem. The NP-hardness of SVP was conjectured in [Bos81] and remained probably the biggest open problem in the area for almost two decades. In a breakthrough paper Ajtai [Ajt98] proved that SVP is NP-hard under randomized reduction. In the recent years the hardness of approximating SVP is being explored, we know that SVP is hard to approximate within *almost* polynomial factor based on reasonable complexity theoretic assumption (see e.g. [HR07]).

Another line of research is to find efficient *exact* algorithm to solve CVP and SVP. The LLL algorithm enables us to solve SVP for constant-dimensional lattices in polynomial time. The fastest known deterministic algorithms to solve SVP or CVP exactly with respect to ℓ_p norm have running time $2^{O(n \log n)}$ ([Kan87], [Bl00]). In a seminal paper [AKS01] Ajtai, Kumar and Sivakumar gave a $2^{O(n)}$ time *randomized* exact algorithm for SVP for ℓ_2 norm. Subsequently, in [AKS02] they gave a $2^{O(n)}$ time randomized algorithm to find a $1+\epsilon$ approximate solution for CVP, for any constant $\epsilon > 0$. Their algorithms are based on a generic sieving procedure.

Another problem recently studied is the *subspace avoiding problem*. Given a k-dimensional subspace $M \subseteq \mathbb{R}^n$ and a full rank integer lattice $\mathcal{L} \subseteq \mathbb{Q}^n$, the *subspace avoiding problem* SAP [BN07], is to find a shortest vector in $\mathcal{L} \setminus M$. If subspace M is zero dimensional

then clearly such SAP instance exactly captures SVP. Blömer and Naewe showed that CVP also reduces to SAP. On the other hand, Micciancio [Mi08] showed that CVP is equivalent to several other lattice problems including shortest independent vector problem (SIVP), successive minima problem (SMP) and subspace avoiding problem(SAP) under deterministic polynomial time rank-preserving reductions. In particular, the reductions in [Mi08] imply a $2^{O(n \log n)}$ time exact algorithm for SAP, SMP and SIVP.

In a breakthrough result Micciancio and Voulgaris [MV10] gave a $2^{O(n)}$ deterministic algorithm to solve many important lattice problems including CVP, SAP, SMP and SVP with respect to ℓ_2 norm. Their algorithm crucially uses the fact that Voronoi cell of a lattice is convex when concerned metric is ℓ_2 norm. For the general ℓ_p norms the Voronoi cell need not be convex. Very recently, via a clever ellipsoid covering technique Dadush, Piekert and Vempala [DPV10] have extended this result to all ℓ_p norms.

Results in this thesis

In the work presented in this thesis we focus on the application of AKS sieving to the problem of finding an exact solution for SAP and CVP with respect to general ℓ_p norms.

We know that SAP is a generalization of both SVP and CVP. When dimension of the input subspace is zero, it exactly captures SVP for which we have exact $2^{O(n)}$ algorithm [AKS01]. So it is natural to explore the complexity of SAP as the dimension of the subspace increases. Given a rank n integer lattice $\mathcal{L}$ and a subspace $M \subset \mathbb{R}^n$ of dimension k we give a $2^{O(n+k \log k)}$ algorithm to solve SAP with respect to ℓ_p norm. This algorithm is based on the AKS sieving. [BN07] also give an algorithm for SAP based on the AKS sieving. Our algorithm performs better, parameterized on the dimension of the subspace because in our analysis we exploit the coset structure of the lattice $\mathcal{L} \cap M$ inside $\mathcal{L}$. This enable us to sample lattice points from a coset of a shortest vector in $\mathcal{L} \setminus M$ and apply packing argument within the coset. As applications of this algorithm we obtain the following results:

- We show that given a full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ there is $2^{O(n)}$ time randomized algorithm to compute linearly independent vectors $v_1, v_2, \ldots, v_i \in \mathcal{L}$ such that $\|v_i\|_p = \lambda_i^p(\mathcal{L})$ if i is $O(\frac{n}{\log n})$, where $\lambda_i^p(\mathcal{L})$ denotes the i^{th} successive minima of

$\mathcal{L}$ with respect to ℓ_p norm. . Given a full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ and $v \in \mathbb{Q}^n$ we also give a $2^{O(n)}$ time algorithm to solve CVP($\mathcal{L}, v$) if the input $(v, \mathcal{L})$ fulfils the promise $d(v, \mathcal{L}) \leq \frac{\sqrt{3}}{2}\lambda_{O(\frac{n}{\log n})}(\mathcal{L})$.

- We show that CVP with respect to ℓ_p norm can be solved in $2^{O(n)}$ time if there is a $2^{O(n)}$ time algorithm to compute a closest vector to v in $\mathcal{L}$ where $v \in \mathbb{Q}^n$, $\mathcal{L} \subset \mathbb{Q}^n$ is a full rank lattice and $v_1, v_2, \ldots, v_n \in \mathcal{L}$ such that $\|v_i\|_p$ is equal to i^{th} successive minima of $\mathcal{L}$ for $i = 1$ to n are given as an additional input to the algorithm. As a consequence, we can assume that successive minimas are given for free as an input to the algorithm for CVP.

- We give a new $2^{O(n+k\log 1/\epsilon)}$ time randomized algorithm to get a $1+\epsilon$ approximate solution for SAP, where n is the rank of the lattice and k is the dimension of the subspace. We get better approximation guarantee than the one in [BN07] parameterized on k.

- We show that the AKS-sieving not only works for all ℓ_p norms, but also for a more general notion of norm specified by a *gauge function* [Si45] and we need only oracle access to the gauge function.

The results presented in this chapter appeared in [AJ08b].

1.2 Algorithmic Problems for metrics on Permutation Groups

Motivated by the generic nature of the AKS-sieving procedure, it is natural to ask whether it can work for similar optimization problems in the other domains. Specifically, we investigate the computational complexity of the two natural problems for metrics on permutation groups given by generating sets (a noncommutative domain).

Given a metric d on the symmetric group S_n, the *weight* of a permutation $\pi \in S_n$ with respect to d is $w_d(\pi) = d(\pi, e)$ where e denotes the identity permutation. Given a permutation group $G = \langle A \rangle \leq S_n$ by a generating set A of permutations, we explore the algorithmic complexity of the *minimum weight problem* (denoted MWP) and the

subgroup distance problem (denoted SDP) for natural permutation metrics. Given a permutation group $G \leq S_n$ by a generating set A and $k \in \mathbb{R}^+$, then for a metric d on permutations, MWP with respect to metric d is to check if $min_{\pi \in G \setminus \{e\}} w_d(\pi) \leq k$. Similarly, given a permutation group G by a set of generators, a permutation $\pi \in G$, and $k \in \mathbb{R}^+$ the subgroup distance problem with respect to a metric d is to check if there exist a permutation $\rho \in G$ such that $d(\pi, \rho) \leq k$. These problems were studied by Cameron et.al. in [BCW06, CW06] and shown to be NP-complete for several natural permutation metrics.

These problems are analogous to the shortest vector problem and the closest vector problem for integer lattices, and to the minimum Hamming weight problem and nearest codeword problem for linear codes. The corresponding problems for lattices and codes are NP-hard, and their approximability is a subject of current intensive study (see e.g. [MG02]). Our primary motivation stems from the fact that lattices and codes are abelian groups, and it is interesting to ask if the upper and lower bound techniques and results for approximability can be extended to arbitrary (nonabelian) permutation groups.

Results in this thesis

We study the complexity of MWP with respect to Hamming and ℓ_∞ metrics. Hamming distance between permutations $\tau, \pi \in S_n$ is defined as $d(\tau, \pi) = |\{i | \tau(i) \neq \pi(i)\}|$. ℓ_∞ distance between τ, π is $d(\tau, \pi) = max_{1 \leq i \leq n} |\tau(i) - \pi(i)|$. MWP is NP-hard with respect to both of these metrics even for abelian permutation groups.

A naive brute-force search algorithm for MWP (which enumerates all the permutations and finds a permutation in G with shortest nonzero norm) can take upto $n!$ steps since $G \leq S_n$ can have up to $n!$ elements. It easily follows that if $G \leq S_n$ is an abelian group then $|G| \leq 2^{O(n)}$, so using classical Schrier-Sims algorithm for finding pointwise stabilizer subgroups of permutation groups ([Lu93]) we can enumerate all the permutations in G and find one with the smallest nonzero norm. This gives a $2^{O(n)}$ algorithm to solve MWP for abelian groups.

More interesting case is that of the nonabelian permutation groups. In the case of Hamming metric we give a deterministic $2^{O(n)}$ time algorithm which is *group theoretic* in

nature. The algorithm is based on the classical Schrier-Sims algorithm. However, the problem for l_∞ metric does not appear amenable to a permutation group-theoretic approach. We give a $2^{O(n)}$ time randomized algorithm for the problem. Interestingly, for this algorithm we are able to adapt ideas from the Ajtai-Kumar-Sivakumar algorithm for the shortest vector problem for integer lattices [AKS01]. Other results presented in the thesis include:

- It is known that SDP is NP-hard([BCW06]) and it easily follows that SDP is hard to approximate within a factor of $\log^{O(1)} n$ unless P=NP. In contrast, we show that SDP for approximation factor more than $n/\log n$ is not NP-hard unless there is an unlikely containment of complexity classes.
- For several permutation metrics, we show that the minimum weight problem is polynomial-time reducible to the subgroup distance problem for *solvable* permutation groups.

These results adapts ideas from the analogous results in the case of integer lattices. The results presented in this chapter appeared in [AJ08a].

1.3 Arithmetic Circuits, Branching Programs and Monomial Algebras

In this part of the thesis we explore the polynomial identity testing problem and certain lower bound questions for arithmetic circuits and algebraic branching programs. Superpolynomial lower bounds for the size of commutative arithmetic circuits or algebraic branching programs for explicit polynomials is one of the most challenging open problem in arithmetic circuit complexity. Lower bounds are known only for some of the special classes of the commutative arithmetic circuits like depth 3 circuits, some restricted classes of depth 4 circuits etc. The general lower bound question is still unsolved despite the efforts of several researchers.

In the noncommutative case the question is better understood. Nisan in the pioneering paper [N91] studied the lower bounds for the noncommutative computation. Using

a rank argument Nisan showed that the Permanent and the Determinant polynomials in the *free* noncommutative ring $\mathbb{F}\{x_{11}, \cdots, x_{nn}\}$ require exponential size noncommutative formulas (and the noncommutative algebraic branching programs). Chien and Sinclair [CS04] explored the same question over other noncommutative algebras. They refined Nisan's rank argument to show an exponential size lower bounds for formulas computing the Permanent or the Determinant over the algebra of 2×2 matrices over $\mathbb{F}$, the quaternion algebra, and several other interesting examples.

In a similar spirit as [CS04], in the work presented in this thesis we explore the lower bound question over other noncommutative algebras. An ideal I of the noncommutative polynomial ring $\mathbb{F}\{x_1, \ldots, x_n\}$ (which we denote by $\mathbb{F}\{X\}$ when the set of indeterminates is clear from the context) is a subring that is closed under both left and right multiplication by the ring elements. The circuit complexity of the polynomial f in the quotient algebra $\mathbb{F}\{X\}/I$ is $C_I(f) = \min_{g \in I} C(f+g)$ where for $h \in \mathbb{F}\{X\}$, $C(h)$ is the circuit complexity of h over free noncommutative algebra $\mathbb{F}\{X\}$. We can define the algebraic branching program complexity of a polynomial over $\mathbb{F}\{X\}/I$ analogously. We study the question of proving lower bound on the arithmetic circuit complexity and the algebraic branching program complexity of explicit polynomials over quotient algebra $\mathbb{F}\{X\}/I$ where the ideal I is given by generating set of polynomials.

If the ideal I is generated by monomials in $\mathbb{F}\{X\}$ the algebra $\mathbb{F}\{X\}/I$ is called as *monomial algebra*. It turns out that the structure of monomial algebras is intimately connected with the automata theory. Next we state the main results in this chapter.

Results in this thesis

- We show that the $n \times n$ Permanent (and Determinant) in the quotient algebra $\mathbb{F}\{x_{11}, x_{12}, \ldots, x_{nn}\}/I$ requires $2^{\Omega(n)}$ size ABPs if the ideal I is generated by $2^{o(n)}$ many monomials. Hence, we can extend Nisan's lower bound argument to noncommutative monomial algebras. Furthermore, the Raz-Shpilka deterministic identity test for noncommutative ABPs [RS05] also carries over to $\mathbb{F}\{X\}/I$.

- In the commutative setting, we prove a $2^{\Omega(n)}$ lower bound for the $n \times n$ Permanent over $\mathbb{Q}[x_{11}, x_{12}, \cdots, x_{nn}]/I$, where the monomial ideal I is generated by

$o(n/\log n)$ monomials. This extends Jerrum and Snir's [JS82] exponential size lower bound result for monotone arithmetic circuits to a similar lower bound result over commutative monomial algebras.

We also study the *Monomial Search Problem*. This is a natural search version of polynomial identity testing: Given a polynomial $f \in \mathbb{F}\{x_1, \ldots, x_n\}$ (or, in the commutative case $f \in \mathbb{F}[x_1, \ldots, x_n]$) of total degree d by an arithmetic circuit or an ABP, the problem is to *find* a nonzero monomial of the polynomial f. We give a randomized NC^2 algorithm for finding a nonzero monomial and its coefficient in both the commutative as well as the noncommutative setting.

The results presented in this chapter appeared in [AJ09].

1.4 Hadamard Product of Polynomials and the Identity Testing Problem

In the work presented in this thesis we introduce and study the Hadamard product of the multivariate polynomials in the free noncommutative polynomial ring $\mathbb{F}\{x_1, x_2, \cdots, x_n\}$. Our definition of the Hadamard Product can be seen as an algebraic generalization of the intersection of the formal languages. Our definition is motivated by the well know Hadamard product of matrices. Hadamard product of matrices of same dimension is simply the entry-wise product.

Suppose $X = \{x_1, x_2, \cdots, x_n\}$ is a set of n noncommuting variables. For a field $\mathbb{F}$ let $\mathbb{F}\{x_1, x_2, \cdots, x_n\}$ denote the free noncommutative polynomial ring over $\mathbb{F}$ generated by the variables in X. We define the Hadamard product of polynomials as follows. Let $f, g \in \mathbb{F}\{X\}$ where $X = \{x_1, x_2, \cdots, x_n\}$. The *Hadamard product* of f and g, denoted $f \circ g$, is the polynomial $f \circ g = \sum_m a_m b_m m$, where $f = \sum_m a_m m$ and $g = \sum_m b_m m$, where the sums index over monomials m.

To see the connection of this definition with that of Hadamard product of two matrices we recall the definition of communication matrices [N91] associated with a degree d homogeneous polynomial $f \in \mathbb{F}\{X\}$. For $k = 1, \ldots, d$ the communication matrix $M_k(f)$

has its rows indexed by degree k monomials and columns by degree $d-k$ monomials and the $(m,m')^{th}$ entry of $M_k(f)$ is the coefficient of mm' in f. It follows easily that Hadamard product of communication matrices associated with two polynomials f and g is same as the communication matrix associated with their Hadamard product (as defined above).

Results in this thesis

We explore the arithmetic circuit and the branching program complexity of the Hadamard product of the polynomials when they are individually given by arithmetic circuits and/or algebraic branching programs. We also study the problem of polynomial identity testing for noncommutative ABPs. Using the results on the Hadamard product of polynomials we give a tight classification for the identity testing problem in case of the field of rationals.

We show that the *noncommutative* branching program complexity of the Hadamard product $f \circ g$ is upper bounded by the product of the branching program sizes for f and g. This upper bound is natural because we know from Nisan's seminal work [N91] that the algebraic branching program (ABP) complexity $B(f)$ is well characterized by the ranks of its "communication" matrices $M_k(f)$, and the rank of Hadamard product $A \circ B$ of two matrices A and B is upper bounded by the product of their ranks. Our proof is constructive: we give a deterministic logspace algorithm for computing an ABP for $f \circ g$.

We apply the above result on the Hadamard Product of two polynomials given by ABPs to tightly classify the identity testing problem for noncommutative ABPs over field of rationals. It is shown by Raz and Shpilka [RS05] that the polynomial identity testing problem for noncommutative ABPs can be solved in deterministic polynomial time. Using result on Hadamard Product of two ABPs, we show that the identity testing problem for noncommutative ABPs over *rationals* is equivalent to the matrix singularity problem under logspace many-one reductions. Matrix singularity problem is to check whether given integer square matrix is singular or not. It is shown in [AO96] that matrix singularity problem is complete for $\mathrm{C_{=}L}$ with respect to logspace many-one reductions when the field is of rational numbers. So our result implies that the identity testing problem in

case of rationals is $\mathrm{C_{=}L}$-complete which is known to be contained in deterministic NC^2.

We show that, the identity testing problem for the noncommutative ABPs over finite field of characteristic p is equivalent to Matrix Singularity problem over field of characteristic p under randomized logspace reduction. Firstly this reduction shows a randomized NC^2 upper bound on the complexity of the problem and it follows from [AO96] that the problem is in randomized $\mathrm{Mod}_p\mathrm{L}$. Using standard amplification techniques we get a $\mathrm{Mod}_p\mathrm{L}/Poly$ upper bound. We also investigate the parallel complexity of the problem. We show that Raz-Shpilka identity test can be parallelized which gives a NC^3 upper bound for the identity testing problem for the non-commutative ABPs over any field.

It turns out that the problem is hard (with respect to logspace many-one reductions) for both NL and $\mathrm{Mod}_p\mathrm{L}$. Hence, it is not likely to be easy to improve the upper bound unconditionally to $\mathrm{Mod}_p\mathrm{L}$ (it would imply that NL is contained in $\mathrm{Mod}_p\mathrm{L}$). Nevertheless it is an interesting question whether we can show deterministic NC^2 upper bound on the identity testing problem for noncommutative ABPs over finite fields?

We explore the expressive power of the Hadamard product of two polynomials when either or both of them given by arithmetic circuit. We show that if either of the two polynomials is given by an ABP the we can efficiently (in logspace) compute an arithmetic circuit for the Hadamard product of the polynomials. But if both the polynomials are give by arithmetic circuits then it is not easy to come up with an efficient algorithm to compute an arithmetic circuit for the Hadamard product of the two polynomials (We show that such an algorithm would imply a non-trivial circuit-size lower bound).

We also consider following identity testing question: Given two polynomials $f, g \in \mathbb{F}\{X\}$ either by an ABP or by an arithmetic circuit check whether $f \circ g$ is identically zero. We show that if both the polynomials are given by arithmetic circuits the problem is coNP-hard even when the circuits are monotone. Whereas, if either of the polynomials is given by an ABP the problem has polynomial time algorithm.

The work presented in this chapter appeared in [AJS09].

2

Sieving Algorithms for Lattice Problems

In this chapter we study the algorithmic complexity of various lattice problems based on the Ajtai-Kumar-Sivakumar(AKS) sieving technique [AKS01].

2.1 Introduction

Lattices are discrete additive subgroups of $\mathbb{R}^n$. Given a set of linearly independent vectors $B = \{b_1, \ldots, b_m\} \subset \mathbb{R}^n$ the lattice $\mathcal{L}$ generated by B is the set of all integer linear combinations of b_i's i.e. $\mathcal{L} = \{\Sigma_{i=1}^{m} \alpha_i b_i | \alpha_1, \alpha_2, \ldots, \alpha_m \in \mathbb{Z}\}$ and B is called as a basis for $\mathcal{L}$. n, m are called rank, dimension of $\mathcal{L}$ respectively. Despite their apparent simplicity, lattices hide a rich combinatorial and algebraic structure which attracted attention of lot of mathematicians through out the last century. Minkowski pioneered the study of integer lattices and christened this new branch of mathematics as *Geometry of Numbers*. It has numerous applications in entire mathematics and in particular in Number Theory. E.g. Minkowski's work simplified theory of units of algebraic number-fields, simplified and extended theory of approximation of irrational numbers.

Two fundamental algorithmic problems concerning integer lattices are the shortest vector problem (SVP) and the closest vector problem(CVP). Given a lattice $\mathcal{L} \subset \mathbb{R}^n$ by a basis, the shortest vector problem (SVP) is to find a shortest non-zero vector in $\mathcal{L}$ with respect to a given metric. Likewise, the closest vector problem (CVP) takes as input a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a vector $v \in \mathbb{R}^n$ and asks for a $u \in \mathcal{L}$ closest to v with respect

to a given metric. Earliest known algorithmic result about integer lattices attributed to Gauss. The result was framed in the language of quadratic forms. For integer lattices, it essentially gives a deterministic polynomial time algorithm for SVP in 2 dimensions. The study of lattices from the computational point of view was marked by a major breakthrough: The LLL algorithm developed by Lenstra, Lenstra and Lovasz [LLL82] which gives an approximate solution for SVP in n dimensions generalizing Gauss' algorithm. Given a rank n integer lattice, in deterministic polynomial time the LLL algorithm outputs a nonzero vector $v \in \mathcal{L}$ whose ℓ_2 norm is guaranteed to be within a $2^{O(n)}$ factor of the norm of a shortest nonzero vector in $\mathcal{L}$. The LLL algorithm can also be used to solve CVP within a $2^{O(n)}$ factor [Bab86].

Despite the relatively poor quality of the approximate solution in the worst case, the LLL algorithm allows us to devise polynomial time algorithms for various problems including polynomial factorization over rationals, breaking knapsack based cryptosystem, solving integer programs in a fixed number of variables etc [LLL82, MG02]. There has been efforts to improve upon the exponential approximation factor in the LLL solution considering its practical importance. Schnorr [Sch94] combined LLL basis reduction technique with Korkine-Zolotarev reduction which gives slightly subexponential factor $2^{O(\frac{n(\log\log n)^2}{\log n})}$.

NP-hardness of the natural decision version of CVP (in any ℓ_p norm) and SVP (in ℓ_∞ norm) was originally proved by van Emde Boas [Bos81]. In fact, it is known that, even finding an approximate solution for CVP is a hard problem (e.g. [ABSS97], [DKS98]). In [DKS98] it is shown that CVP is NP-hard to approximate within approximation factor of $2^{O(\frac{\log n}{\log\log n})}$. Showing NP-hardness for CVP for polynomial approximation factor is an important open problem. The NP-hardness of SVP was conjectured in [Bos81] and remained probably the biggest open problem in the area for almost two decades. In a breakthrough paper Ajtai [Ajt98] proved that SVP is NP-hard under randomized reduction. After this result there were efforts to show that SVP is hard to approximate under some reasonable complexity theoretic assumptions. Best known result in this direction is, for any $\epsilon > 0$ there is no polynomial time algorithm approximating SVP on n-dimensional lattice in ℓ_p norm to within a factor of $2^{O((\log n)^{1-\epsilon})}$ unless NP is contained in $RTIME(2^{(poly(\log n))})$ [HR07].

Another line of research was to find an efficient algorithm to solve CVP and SVP ex-

actly. The LLL algorithm first time enabled us to solve SVP in fixed dimension in polynomial time (the dependency of the running time of the algorithm on the rank of the input lattice is $2^{O(n^2)}$. The fastest known deterministic algorithms to solve SVP or CVP exactly with respect to ℓ_p norm have running time $2^{O(n \log n)}$ ([Kan87], [Bl00]). In a seminal paper [AKS01] Ajtai, Kumar and Sivakumar gave a $2^{O(n)}$ time *randomized* exact algorithm for SVP for ℓ_2 norm. Subsequently, in [AKS02] they gave a $2^{O(n)}$ time randomized algorithm to find a $1+\epsilon$ approximate solution for CVP, for any constant $\epsilon > 0$. Their algorithms are based on a generic sieving procedure. AKS-sieving combined with Schnorr's reduction technique based on the LLL and Korkine-Zolotarev basis reduction gives a randomized polynomial time algorithm to solve SVP within an approximation factor of $2^{O(\frac{n(\log \log n)}{\log n})}$.

Other well studied lattice problems include successive minima problem (SMP) and shortest independent vector problem(SIVP). For $1 \leq i \leq n$, the i^{th} *successive minima* of lattice $\mathcal{L}$, $\lambda_i(\mathcal{L})$ is defined as the smallest r such that a ball of radius r around origin contains at least i linearly independent lattice vectors. The successive minimas $\lambda_i(\mathcal{L})$ are important lattice parameters. *Successive minima problem SMP* of finding, for a given lattice $\mathcal{L}$, n linearly independent vectors $v_1, v_2, \ldots, v_n \in \mathcal{L}$ such that $\|v_i\|$ is at most $\lambda_i(\mathcal{L})$. This problem clearly subsumes *shortest independent vector problem SIVP* where given a lattice $\mathcal{L}$ one wants to find linearly independent vectors $v_1, v_2, \ldots, v_n \in \mathcal{L}$ such that $\|v_i\| \leq \lambda_n(\mathcal{L})$.

Another problem recently studied is the *subspace avoiding problem*. Given a k dimensional subspace $M \subseteq \mathbb{R}^n$ and a full rank integer lattice $\mathcal{L} \subseteq \mathbb{Q}^n$, the *subspace avoiding problem* SAP [BN07], is to find a shortest vector in $\mathcal{L} \setminus M$. If subspace M is zero dimensional then clearly such SAP instance exactly captures SVP. Blömer and Naewe showed that CVP also reduces to SAP. On the other hand, Micciancio [Mi08] showed that CVP is equivalent to several other lattice problems including shortest independent vector problem, successive minima problem and subspace avoiding problem under deterministic polynomial time rank-preserving reductions. In particular, the reductions in [Mi08] imply a $2^{O(n \log n)}$ time exact algorithm for SAP, SMP and SIVP. [BN07] gives a $2^{O(n)}$ time algorithm to find a $1+\epsilon$ approximate solution for these problems with respect to ℓ_p norm for a constant $\epsilon > 0$.

Recently, in a breakthrough result Micciancio and Voulgaris [MV10] gave a $2^{O(n)}$ deter-

ministic algorithm to solve many important lattice problems including CVP, SAP, SMP and SVP with respect to ℓ_2 norm. Their algorithm crucially uses the fact that Voronoi cell of a lattice is convex when concerned metric is ℓ_2 norm. For the general ℓ_p norms the Voronoi cell need not be convex. So their algorithm doesn't directly generalize for other metrics. Very recently, via an ellipsoid covering technique Dadush, Piekert and Vempala [DPV10] have extended results from [MV10] to all ℓ_p norms.

Results in this chapter

The problem of getting an exact $2^{O(n)}$ algorithm for CVP, SAP and SIVP remains open for general ℓ_p norms. Whereas all of these problems have $2^{O(n)}$ time algorithms which give a $1+\epsilon$ approximate solutions to the problems for any constant $\epsilon > 0$ with respect to general ℓ_p norms. In this chapter we make an effort to understand the complexity of finding an exact solution for SAP and CVP with respect to general ℓ_p norms.

We know that SAP is a generalization of both SVP and CVP. When dimension of the input subspace is zero, it exactly captures SVP for which we have exact $2^{O(n)}$ algorithm [AKS01]. But for the general instance of SAP we only have a $2^{O(n \log n)}$ exact algorithm. So it is natural to explore the complexity of SAP as the dimension of the subspace increases. Given a rank n integer lattice $\mathcal{L}$ and a subspace $M \subset \mathbb{R}^n$ of dimension k we give a $2^{O(n+k \log k)}$ algorithm to solve SAP with respect to ℓ_p norm. This algorithm is based on the AKS sieving. [BN07] also give an algorithm for SAP base on the AKS sieving. Our algorithm performs better, parameterized on the dimension of the subspace because in our analysis we exploit the coset structure of the lattice $\mathcal{L} \cap M$ inside $\mathcal{L}$. This enable us to sample lattice points from a coset of a shortest vector in $\mathcal{L} \setminus M$ and apply packing argument within the coset. As applications of this algorithm we obtain the following results:

- We show that given a full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ there is $2^{O(n)}$ time randomized algorithm to compute linearly independent vectors $v_1, v_2, \ldots, v_i \in \mathcal{L}$ such that $\|v_i\|_p = \lambda_i^p(\mathcal{L})$ if i is $O(\frac{n}{\log n})$, where $\lambda_i^p(\mathcal{L})$ denotes the i^{th} successive minima of $\mathcal{L}$ with respect to ℓ_p norm. . Given a full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ and $v \in \mathbb{Q}^n$ we also give a $2^{O(n)}$ time algorithm to solve CVP($\mathcal{L}, v$) if the input $(v, \mathcal{L})$ fulfils the promise $d(v, \mathcal{L}) \leq \frac{\sqrt{3}}{2} \lambda_{O(\frac{n}{\log n})}(\mathcal{L})$.

- We show that CVP with respect to ℓ_p norm can be solved in $2^{O(n)}$ time if there is a $2^{O(n)}$ time algorithm to compute a closest vector to v in $\mathcal{L}$ where $v \in \mathbb{Q}^n$, $\mathcal{L} \subset \mathbb{Q}^n$ is a full rank lattice and $v_1, v_2, \ldots, v_n \in \mathcal{L}$ such that $\|v_i\|_p$ is equal to i^{th} successive minima of $\mathcal{L}$ for $i = 1$ to n are given as an additional input to the algorithm. As a consequence, we can assume that successive minimas are given for free as an input to the algorithm for CVP.

- We give a new $2^{O(n+k\log 1/\epsilon)}$ time randomized algorithm to get a $1+\epsilon$ approximate solution for SAP, where n is the rank of the lattice and k is the dimension of the subspace. We get better approximation guarantee than the one in [BN07] parameterized on k.

- We show that the AKS-sieving not only works for all ℓ_p norms, but also for a more general notion of norm specified by a *gauge function* [Si45] and we need only oracle access to the gauge function.

2.2 Preliminaries

Integer Lattices:

A lattice $\mathcal{L}$ is a discrete additive subgroup of R^n, n is called dimension of the lattice. For algorithmic purposes we can assume that $\mathcal{L} \subseteq \mathbb{Q}^n$, and even in some cases $\mathcal{L} \subseteq \mathbb{Z}^n$. A lattice is usually specified by a basis $B = \{b_1, \cdots, b_m\}$, where $b_i \in \mathbb{Q}^n$ and b_i's are linearly independent. m is called the rank of the lattice. If the rank is n the lattice is said to be a *full rank* lattice. Although most results in the paper hold for general lattices, for convenience henceforth we consider only full-rank lattices.

For a full rank lattice $\mathcal{L}$ generated by basis $B = \{b_1, \ldots, b_n \in \mathbb{Q}^n\}$ let us denote the fundamental parallelepiped of lattice $\mathcal{L}$ associated with the basis B by $\mathcal{P}(B) = \{\sum_i x_i b_i | x_i \in \mathbb{R}, 0 \leq x_i < 1 \text{for } 1 \leq i \leq n\}$. For any lattice basis B and a point $x \in \mathbb{R}^n$ there is unique vector $y \in \mathcal{P}(B)$ such that $y - x \in \mathcal{L}$. This vector is denoted by $y = x(\mod \mathcal{L}(B))$.

For $x \in \mathbb{Q}^n$ let $\mathrm{size(x)}$ denote the number of bits for the standard binary representation as an n-tuple of rationals. Let $\mathrm{size}(\mathcal{L})$ denote $\sum_i \mathrm{size(b_i)}$.

Gauge functions:

Definition 2.1 (gauge function) [Si45] *A function* $f : \mathbb{R}^n \to \mathbb{R}$ *is called a* gauge function *if it satisfies following properties:*

1. $f(x) > 0$ *for all* $x \in \mathbb{R}^n \setminus \{0\}$ *and* $f(x) = 0$ *if* $x = 0$.
2. $f(\lambda x) = \lambda f(x)$ *for all* $x \in \mathbb{R}^n$ *and* $\lambda \in \mathbb{R}$.
3. $f(x + y) \leq f(x) + f(y)$ *for all* $x, y \in \mathbb{R}^n$.

For $v \in \mathbb{R}^n$ we denote $f(v)$ by $\|v\|_f$ and call it norm of v with respect to gauge function f. It is well-known that any l_p norm satisfies all the above properties. Thus gauge functions generalize the usual l_p norms. We denote the ℓ_p norm of vector v by $\|v\|_p$. A gauge function f defines a natural metric d_f on $\mathbb{R}^n$ by setting $d_f(x, y) = f(x - y)$ for $x, y \in \mathbb{R}^n$. For $x \in \mathbb{R}^n$ and $r > 0$, let $B_f(x, r)$ denote the f-ball of radius r with center x with respect to the gauge function f, defined as $B_f(x, r) = \{y \in \mathbb{R}^n | f(x - y) \leq r\}$. We denote the metric ball of radius r around point $x \in \mathbb{R}^n$ with respect to usual l_p norm $p >\geq 1$ by $B_p(x, r)$, similarly the metric ball with respect to ℓ_∞ norm is denoted by $B_\infty(x, r)$. Unless specified otherwise we always consider balls in $\mathbb{R}^n$. for a measurable set $S \subset \mathbb{R}^n$, $Vol(S)$ denotes the volume of S. The next well-known proposition characterizes the class of all gauge functions.

Proposition 2.2 [Si45] *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be any gauge function then a unit radius ball around origin with respect to* f *is a* n *dimensional bounded O-symmetric convex body. Conversely for any* n *dimensional bounded O-symmetric convex body* C*, there is a gauge function* $f : \mathbb{R}^n \to \mathbb{R}$ *such that* $B_f(0, 1) = C$.

Given an f-ball of radius r around origin with respect to a gauge function f, from the Proposition 2.2 it follows that $B_f(0, r)$ is an O-symmetric convex body. Next we prove a useful fact about volume of f-balls.

Proposition 2.3 *Let* $M \subset \mathbb{R}^n$ *be a subspace of dimension* $k < n$ *and* $f : \mathbb{R}^n \to \mathbb{R}$ *be a gauge function then for any constant* c*,* $Vol(B_f(0, cr) \cap M) = c^k Vol(B_f(0, r) \cap M)$.

Proof Let $b_1, \ldots, b_k$ be an orthonormal basis of M. Let $T : \mathbb{R}^n \to \mathbb{R}^k$ be an orthogonal transformation such that $T(b_i) = e_i$, for $1 \leq i \leq k$. Let $U = B_f(0, cr) \cap M$ and $V = B_f(0, r) \cap M$. So $T(U), T(V) \subset \mathbb{R}^k$. It is easy to see that $T(U)$ and $T(V)$ are O-symmetric bounded nondegenerate convex bodies. So we have

$$Vol(T(U)) = \int_{\{x=(x_1,\ldots x_k)\in T(U)\}} dx_1 \ldots dx_k$$

Let $x = cy$, as T is a linear transformation we have,

$$Vol(T(U)) = \int_{\{y=(y_1,\ldots y_k)\in T(\frac{1}{c}U)\}} c^k dx_1 \ldots dx_k$$

Since $U = cV$ this implies $Vol(T(U)) = c^k Vol(T(V))$. Since T is orthogonal transformation it preserves volume. So we get $Vol(U) = Vol(T(U)) = c^k Vol(T(V)) = c^k Vol(V)$. ∎

For details on gauge functions and lattice theory we refer to [Si45].

For our algorithms we need only oracle access to the gauge function. Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a gauge function. By an *oracle* access to the gauge function f we mean the following: the algorithm is given as an input a black box associated with f. The algorithm can ask queries to the blackbox. On query $x \in \mathbb{R}^n$ the black box returns $f(x)$ in time $O(1)$.

We now place a natural restriction on the gauge functions f that we consider.

Definition 2.4 *A gauge function $f : \mathbb{R}^n \to \mathbb{R}$ given by an oracle is called a* nice gauge function *if there is a polynomial $p(n)$ such that $B_2(0, 2^{-p(n)}) \subseteq B_f(0, 1) \subseteq B_2(0, 2^{p(n)})$, i.e. there exists a Euclidean sphere of radius $2^{-p(n)}$ inside the convex body $B_f(0, 1)$, and $B_f(0, 1)$ is contained inside a Euclidean sphere of radius $2^{p(n)}$.*

If f is a nice gauge function and $v \in \mathbb{Q}^n$ we have size($f(v)$)=poly(n,size(v)). More importantly, for a nice gauge function f and $r \in \mathbb{R}^+$ we can sample points from convex body $B_f(0, r)$ almost uniformly at random in polynomial time using Dyer-Frieze-Kannan algorithm [DFK91].

Following easy claim shows that usual l_p norms $p \geq 1$ define nice gauge functions.

Claim 1 *For any $p \geq 1$, $B_2(0, \frac{1}{\sqrt{n}}) \subseteq B_p(0,1) \subseteq B_2(0, \sqrt{n})$.*

Proof Let $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$. Clearly, $(max_{i=1}^n |x_i|)^p \leq \sum_{i=1}^n |x_i|^p$. So it follows that $B_p(0,1) \subseteq B_\infty(0,1)$. We have $(\sum_{i=1}^n |x_i|)^p \geq \sum_{i=1}^n |x_i|^p$, this implies $B_1(0,1) \subseteq B_p(0,1)$. So we have $B_1(0,1) \subseteq B_p(0,1) \subseteq B_\infty(0,1)$. Clearly, $B_\infty(0,1) \subseteq B_2(0, \sqrt{n})$. By Cauchy-Schwartz inequality it follows that $B_2(0, \frac{1}{\sqrt{n}}) \subseteq B_1(0,1)$. ∎

The i^{th} successive minima of a lattice $\mathcal{L}$ with respect to gauge function f is smallest $r > 0$ such that $B_f(0,r)$ contains at least i linearly independent lattice vectors. It is denoted by $\lambda_i^f(\mathcal{L})$. If the concerned norm is ℓ_p norm we denote the i^{th} successive minima by $\lambda_i^p(\mathcal{L})$.

We consider lattice problems with respect to nice gauge functions. Let $\mathcal{L}$ be a lattice with basis $\{b_1, b_2, \ldots, b_n\}$ and f be a nice gauge function. Suppose B is a full rank $n \times n$ matrix with columns $b_1, b_2, \ldots, b_n$. Note that the linear transformation B^{-1} maps lattice $\mathcal{L}$ isomorphically to the standard lattice $\mathbb{Z}^n$. Furthermore, it is easy to see that the set $C = B^{-1}(B_f(0,1))$ is an O-symmetric convex body. So, by Proposition 2.2, $C = B_g(0,1)$ for some gauge function g. As f is a nice gauge function, simple calculations shows that g is also a nice gauge function.

Thus, our algorithms that work for nice gauge functions can be stated for the the standard lattice $\mathbb{Z}^n$ and a nice gauge function g. However, some of our results hold only for ℓ_p norms. Thus, to keep uniformity we allow our algorithms to take arbitrary lattices as input even when the metric is give by a nice gauge function.

Lattice problems: Next we define the important lattice problems which we are going to study in this chapter. All the problems can be considered with respect general norms (associated with gauge functions). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a gauge function.

Shortest Vector Problem (SVP): Given a lattice $\mathcal{L} \subset \mathbb{R}^n$ find nonzero $v \in L$ of least possible f-norm.

Closest Vector Problem(CVP): Given a lattice $\mathcal{L} \subset \mathbb{R}^n$ and $u \in \mathbb{R}^n$ find $u \in L$ such that $\|v - u\|_f$ is least possible. Given a lattice $\mathcal{L} \subset \mathbb{R}^n$ find

Successive Minima Problem(SMP): Given a lattice $\mathcal{L} \subset \mathbb{R}^n$ find linearly independent vectors $v_1, \ldots, v_n \in \mathcal{L}$ such that $\|v_i\| \leq \lambda_i^f(\mathcal{L})$.

Subspace Avoiding Problem(SAP): Given a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a subspace $M \subset \mathbb{R}^n$ of dimension $k < n$ find a vector $v \in \mathcal{L} \setminus M$ with least possible norm.

2.3 A Sieving Algorithm for SAP

Before explaining our algorithm for SAP using sieving method, first we briefly describe basic idea behind the AKS sieving algorithm for SVP.

2.3.1 AKS Sieving

Basic idea of the AKS algorithm for SVP is as follows. Given a basis of the lattice, Choose $2^{O(n)}$ random lattice points (almost) uniformly from a sufficiently large parallelepiped and then perturb these lattice points by small quantities using certain distribution. Then apply "sieving" procedure to this set of points. The basic property of this stage is that given a set of perturbed lattice vector with a bound on the longest vector in the set, it outputs a new set of perturbed lattice vectors such that the longest vector is now almost half of the original length while the size of the set doesn't shrink much. This procedure is applied successively until we obtain a set of vectors in a ball of constant radius. Finally argue using the properties of the distribution of the perturbations argue that there is a reasonable chance to find a shortest nonzero vector (and its nearest neighbors) by this process. Crucial technical contribution of AKS algorithm is the idea of adding perturbations to the lattice points, which enable them to argue about the output distribution of the algorithm in a convenient manner.

As explained in the introduction Blömer and Naewe introduced Subspace Avoiding Problem (SAP) in which given an integer lattice $\mathcal{L}$ of rank n and a subspace M of dimension k goal is to find a shortest vector in $\mathcal{L} \setminus M$. They used AKS sieving technique to give a randomized $2^{n \log \frac{1}{\epsilon}}$ time algorithm to find a lattice point $u \in \mathcal{L} \setminus M$ such that $\|v - u\| \leq \epsilon$ where v is a shortest vector in $\mathcal{L} \setminus M$. We also use AKS sieving to give an approximation algorithm for SAP. We give a $2^{n+k \log \frac{1}{\epsilon}}$ time randomized algorithm

which outputs a set of lattice points T such that there is a vector $u \in T$, $u \in \mathcal{L} \setminus M$ and $\|v - u\| \leq \epsilon$ where v is a shortest vector in $\mathcal{L} \setminus M$, moreover v and u lie in a same coset of $\mathcal{L} \cap M$ inside $\mathcal{L}$, i.e. $v - u \in \mathcal{L} \cap M$. The fact that v and u lie in the same coset of $\mathcal{L} \cap M$ is crucial to our applications and leads to a $2^{O(n)}$ time algorithm for restricted versions of various lattice problem. Moreover, our approximation algorithm for SAP works with respect to any nice gauge function as a norm.

2.3.2 Our sieving algorithm for SAP

We give an intuitive outline of our approximation algorithm: Our analysis of AKS sieving will use the fact that the sub-lattice $\mathcal{L} \cap M$ of $\mathcal{L}$ is of rank k. We will use the AKS sieving procedure to argue that we can sample $2^{O(n+k\log(1/\epsilon))}$ points from *some* coset of $\mathcal{L} \cap M$ in $2^{O(n+k\log(1/\epsilon))}$ time. We can then apply a packing argument in the coset (which is only k-dimensional) to obtain points in the coset that are close to each other. Then, with a standard argument following the original AKS result [AKS01] we can conclude that their differences will contain a good approximation to a shortest vector $v \in \mathcal{L} \setminus M$ which lies in the same coset of v.

Let $\mathcal{L} \subset \mathbb{R}^n$ be a rank n input lattice with basis $b_1, \ldots, b_n$ and $M \subset \mathbb{R}^n$ is a given subspace. let f be a nice gauge function with respect to which we want to solve SAP. Let $s = \text{size}(\mathcal{L}, \text{M})$ denote the input size (which is the total number of bits required to represent the vectors b_i's and the basis vectors for M). We denote a length of a shortest vector in $\mathcal{L} \setminus M$ by $sh(\mathcal{L}\ M)$. Let v be a shortest vector in $\mathcal{L} \setminus M$.

Claim 2 *Given an algorithm A to solve SAP $(\mathcal{N}, M)$ w.r.t. nice gauge function f with a promise that $2 \leq sh(\mathcal{N}, M) < 3$ in time t, then there is an algorithm to solve a general instance of SAP $(\mathcal{L}, M)$ of size s, where $\mathcal{L}$ is generated by a basis $B = \{b_1, \ldots, b_n\}$ w.r.t. the gauge function f, in time polynomial in t, s, n.*

Proof Let v be a shortest vector in $\mathcal{L} \setminus M$. Clearly $f(v) \leq max_i f(b_i)$. This implies $f(v) \leq 2^{p(n)} max_i \|b_i\|_2$ for a polynomial p as f is a nice gauge function. Clearly $size(max_i \|b_i\|_2)$ is a polynomial in s. This implies $2^{-q(s,n)} \leq f(v) \leq 2^{q(s,n)}$ for a polynomial q. So we apply algorithm A to solve instance of SAP $(\mathcal{N}, M)$ where N is

generated by basis $2^i B$ for $-q \leq i \leq q$ and pick the best answer among these. Clearly for one of the instance $2 \leq sh(\mathcal{N}, M) \leq 3$ for which A will give a correct answer. Thus proving the claim. ∎

So without loss of generality we will assume that for a given lattice $\mathcal{L}$ and a subspace M we have $2 \leq sh(\mathcal{L}, M) < 3$.

Next we describe sieving procedure due to [AKS01] for any gauge function, analyze its running time and explain its key properties.

Lemma 2.5 (Sieving Procedure) *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be any gauge function. Then there is an algorithm that takes as input a finite set of points $\{v_1, v_2, v_3, \ldots, v_N\} \subseteq B_f(0, r)$, and in $N^{O(1)}$ time it outputs a subset of indices $S \subset [N]$ of size at most 5^n and a mapping $\phi : [N] \to S$ such that for each $i \in [N]$, $f(v_i - v_{\phi(i)}) \leq r/2$.*

Proof The sieving procedure is exactly as described in O. Regev's lecture notes [Re]. The sieving procedure is based on a simple greedy strategy. We start with $S = \emptyset$ and run the following step for all elements $v_i, 1 \leq i \leq N$. If there is a $j \in S$ such that $f(v_i - v_j) \leq r/2$ then set $\phi(i) = j$ for a smallest j possible. If $f(v_i - v_j) > r/2$ for all $j \in S$ include i in the set S, set $\phi(i) = i$ and increment i. After completion, for all $i \in [N]$ we will have $f(v_i - v_{\phi i}) \leq r/2$. The bound on $|S|$ follows from a packing argument combined with the fact that $\text{vol}(B_f(0, cr)) = c^n \text{vol}(B_f(0, r))$ for any $r > 0$ and a constant $c > 0$. More precisely, for any two points $v_i, v_j \in S$ we have $f(v_i - v_j) > r/2$. Thus, all the convex bodies $B_f(v_i, r/4)$ for $v_i \in S$ are mutually disjoint and are contained in $B_f(0, r + r/4)$. Also note that $\text{vol}(B_f(0, dr)) = d^n \text{vol}(B_f(0, r))$ for any constant $d > 0$. It follows that $5^n \text{vol}(B_f(v_i, r/4)) \geq \text{vol}(B_f(0, r + r/4))$. Hence, $|S| \leq 5^n$. ∎

Before describing the Algorithm for SAP we note few facts about the lattice $\mathcal{L} \cap M$. It follows from standard lattice theory that $\mathcal{L} \cap M$ is a rank k lattice which is a subgroup of $\mathcal{L}$. In fact we can compute a basis for the rank k lattice $\mathcal{L} \cap M$ in polynomial time.

Lemma 2.6 [Mi08, Lemma 1] *There is a polynomial time algorithm which on input a lattice $\mathcal{L} \subset \mathbb{Q}^n$ and a subspace $M \subset \mathbb{R}^n$ of dimension $k < n$ outputs a basis for rank k lattice $\mathcal{L} \cap M$.*

Algorithm 1

Input: A lattice $\mathcal{L}$ generated by a basis $B = \{b_1, \ldots, b_n\}$ and a basis for a subspace $M \subset \mathbb{R}^n$ of dimension $k < n$ with promise that $2 \leq sh(\mathcal{L}, M) < 3$ and a real number $\epsilon > 0$.

Output: A set $T = \{(x_1, y_1), \ldots, (x_t, y_t)\}$ such that $x_i, y_i \in \mathbb{R}^n$, $y_i - x_i \in \mathcal{L}$ for $1 \leq i \leq t$ and there exists $1 \leq i, j \leq t$ such that the lattice point $u = (y_i - x_i) - (y_j - x_j)$ lie in the same coset of v (i.e. $v - u \in \mathcal{L} \cap M$) and $f(v - u) \leq \epsilon$ where v is a shortest vector in $\mathcal{L} \setminus M$ w.r.t. f-norm.

1. Let $R' = n \cdot max_i \|b_i\|_f$. Choose $N = 2^{c \cdot (n + k \log(1/\epsilon))} \cdot \log R'$ points $x_1, \ldots x_N$ from $B_f(0, 2)$ uniformly at random and compute $y_i = x_i (\mod \mathcal{L}(B))$ for $1 \leq i \leq N$.
2. Let $P = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and $R = R'$.
3. **While** $R > 8$ **do**
   (a) Apply the sieving procedure from Lemma 2.5 to the set $\{y_1, \ldots, y_N)$ and let $S \subset [N], |S| < 5^n$ be the set output by the sieving procedure and $\phi : [N] \to S$ be the mapping such that $f(y_i - y_{\phi(i)}) \leq R/2$.
   (b) Remove all pairs (x_i, y_i) from P corresponding to $i \in S$.
   (c) Replace each pair (x_i, y_i) for $i \notin S$ in P with $(x_i, y_i - (y_{\phi(i)} - x_{\phi(i)}))$.
   (d) set $R = R/2 + 2$.
4. Output set P

Claim 3 *Following invariants are maintained at the beginning of the While loop.*

- *For all* $(x_i, y_i) \in P$, $y_i - x_i \in \mathcal{L}$.
- *For all* $(x_i, y_i) \in P$, $\|y_i\|_f \leq R$.

Proof As $y_i = x_i (\mod \mathcal{L}(B))$, $y_i \in \mathcal{P}(\mathcal{L}(B))$ for $1 \leq i \leq N$ so $\|y_i\| \leq R'$ for $i \in [N]$. So is clear that both the invariants are true when the While loop is invoked first

time. We are updating the tuple (x_i, y_i) by $(x_i, y_i - (y_{\phi(i)} - x_{\phi(i)}))$ in the step (c) of the While loop. Clearly $y_i - (y_{\phi(i)} - x_{\phi(i)}) - x_i \in \mathcal{L}$ so the first invariant is maintained at the beginning of the While loop. From Lemma 2.5 it follows that $\|y_{\phi(i)} - y_i\|_f \leq R/2$ hence $\|y_i - (y_{\phi(i)} - x_{\phi(i)})\|_f \leq R/2 + 2$ as defined in the step (d) of the While loop. Thus both the invariants are maintained at the beginning of the While loop. ∎

Now we show correctness of the Algorithm and analyze its time complexity.

Theorem 2.7 *Let $\mathcal{L} \subset \mathbb{Q}^n$ is a full rank lattice and let $v \in \mathcal{L} \backslash M$ such that $2 \leq f(v) \leq 3$ for a given gauge function f and $f(v) = sh(\mathcal{L}, M)$. Let $\epsilon > 0$ is an arbitrary constant. Then there is a randomized algorithm that in time $2^{O(n+klog(1/\epsilon))}$.Poly$(\text{size}(\mathcal{L}))$ computes a set P of pairs (x_i, y_i) such that $y_i - x_i \in \mathcal{L}$ and probability $1 - 2^{-O(n)}$ there is a pair $(x_i, y_i), (x_j, y_j) \in P$ such that the lattice point $u = (y_i - x_i) - (y_j - x_j)$ and v lie in the same coset of $\mathcal{L} \cap M$ inside $\mathcal{L}$. (i.e. $v - u \in \mathcal{L} \cap M$ and $f(u - v) \leq \epsilon$.*

Proof First we analyze the time complexity of the algorithm.

From Claim 3 it follows that the While loop is executed at most $O(\log(R')$ times, since f is a nice gauge function, it implies that the While loop is executed at most $poly(n, s)$ times. In the step (1) of the algorithm we are sampling $N = 2^{c(n+k\log(1/\epsilon))} \cdot \log R'$ many vectors uniformly at random from $B_f(0, 2)$. Since f is a nice gauge function this task can be accomplished in time $2^{O(n+k\log(1/\epsilon))}$ using the Dyre-Kannan-Frieze [DFK91] algorithm. Hence the overall running time of the algorithm is $2^{O(n+k\log(1/\epsilon))}$.

Note that from Lemma 2.5 it follows that in each iteration of the while loop at most 5^n pairs are removed and from Claim 3 it follows that the While loop is executed at most $O(\log R')$ times. So by choosing constant c large enough in the first step of the algorithm we can ensure that $|P| \geq 2^{c'(n+k\log(1/\epsilon))}$ for an arbitrary constant c', where P is the set of the pairs (x_i, y_i) obtained after execution of the algorithm.From the Claim 3 it follows that $f(y_i - x_i) \leq 8$ and $y_i - x_i \in \mathcal{L}$ for all $(x_i, y_i) \in P$.

The proof of correctness of the algorithm involve three main parts. First we will argue that P contains "large" number of pairs (x_i, y_i)'s such that all $(y_i - x_i)$'s lie in the same coset of $\mathcal{L} \cap M$ inside $\mathcal{L}$. Next we will apply packing argument in the coset to argue that there exist i, j such that $(x_i, y_i), (x_j, y_j) \in P$ such that $w = y_i - x_i$ and $r = y_j - x_j$ lie in

the same coset of $\mathcal{L} \cap M$ moreover $f(w, r) \leq \epsilon$. Finally we argue that in fact with good probability we can find pairs $(x_i, y_i), (x_j, y_j) \in P$ such that $u = (y_i - x_i) - (y_j - x_j)$ and v lie in same coset of $\mathcal{L} \cap M$ and $f(v - u) \leq \epsilon$ using property of distribution of the x_i's chosen in the first step of the algorithm.

Coset sampling:

We define an equivalence relation on pairs in P. Let $(x_i, y_i) \sim (x_j, y_j)$ if $(y_i - x_i) - (y_j - x_j) \in \mathcal{L} \cap M$ i.e. $y_i - x_i$ and $y_j - x_j$ lie in the same coset of $\mathcal{L} \cap M$. Let $Z_1, \ldots Z_m$ be equivalence classes under $\sim$, i.e. $Z_i \cap Z_j = \phi$ for $i \neq j$ and $Z_1 \cup \ldots \cup Z_m = P$.

Claim 4 (Coset sampling) *By choosing constant c' large enough we can ensure that there is an index t, $1 \leq t \leq m$ such that $|Z_t| \geq 2^{c_2(n+k\log(1/\epsilon))}$ for any constant c_2.*

Proof Let $w_1, \ldots, w_m \in \mathcal{L}$ such that $w_i = y_i - x_i$ for $(x_i, y_i) \in Z_i$ for $1 \leq i \leq m$. Clearly w_i's lie in the different cosets of $\mathcal{L}$ inside $\mathcal{L}$. So $w_i - w_j \notin \mathcal{L} \cap M$. Since we have a promise that $2 \leq sh(\mathcal{L}, M) \leq 3$ it implies $f(w_i - w_j) \geq 2$ for $i \neq j$ i.e. $B_f(w_i, 1)$ are disjoint for $1 \leq i \leq m$. Since for all pairs $(x_i, y_i) \in P$ we have $f(y_i - x_i) \leq 8$ so we have $f(w_i) \leq 8$. So points $w_i, \ldots, w_m$ lie in a boll of radius 8 and ball of radius 1 around them are disjoint. So using packing argument we have $m \leq 9^n$. Note that $Z_1 \cup Z_2 \ldots Z_m = P$ and $|P| \geq 2^{c'(n+k\log(1/\epsilon))}$. So by choosing c' appropriately large we can ensure that there exist t, $1 \leq t \leq m$ and $|Z_t| \geq 2^{c_2(n+k\log(1/\epsilon))}$ for any constant c_2. ∎

By renumbering the indices assume that $Z_t = \{(x_1, y_1), \ldots, (x_q, y_q)\}$, $q \geq 2^{c_2(n+k\log(1/\epsilon))}$. Let $\beta_i = y_i - x_i$ for $(x_i, y_i) \in Z_t$. So all β_i's lie in the same coset $(\mathcal{L} \cap M) + v_\ell$ for a lattice point v_ℓ.

Packing argument in the coset:

Next we will argue that there exists pairs $(x_i, y_i), (x_j, y_j) \in Z_t \subset P$ such that $f((y_i - x_i) - (y_j - x_j)) \leq \epsilon$.

Claim 5 (Packing argument) *By choosing the constant c_2 large enough we can ensure that there exists $(x_i, y_i), (x_j, y_j) \in Z_t, i \neq j$ such that $f(\beta_i - \beta_j) \leq \epsilon$.*

Proof To the contrary suppose that for all $(x_i, y_i), (x_j, y_j) \in Z_t, i \neq j$ $f(\beta_i - \beta_j) \geq \epsilon$ i.e. f-balls of radius $\epsilon/2$ around β_i's $B_f(\beta_i, \epsilon/2)$ are all disjoint. Since $f(\beta_i) \leq 8$ we also have $f(\beta_i - \beta_j) \leq 16$ for $i, j \in [q]$. Let $\gamma_i = \beta_i - v_\ell \in \mathcal{L} \cap M \subset M$ for $i = 1$ to q. It is clear that $f(\gamma_i - \gamma_j) = f(\beta_i - \beta_j) \leq 16$ for $i, j \in [q]$. So $\gamma_1, \ldots, \gamma_q \in M$ lie inside a f-ball of radius 16 and $\epsilon/2$ radius f-balls around γ_i's are disjoint. From Proposition 2.3 and a packing argument it follows that $q = |Z_t| \leq \frac{(16+\epsilon/2)^k}{(\epsilon/2)^k} = 2^{g(k \log(1/\epsilon))}$ for a constant g. This is a contradiction since choosing c_2 large enough we can ensure that $|Z_t| \geq 2^{c_2(n+k\log(1/\epsilon))} > 2^{g(k\log(1/\epsilon))}$. ∎

Argument with modified distribution:

We have argued in the Claim 4 and Claim 5 that P contains pairs $(x_i, y_i), (x_j, y_j)$ such that $f((y_i - x_i) - (y_j - x_j)) \leq \epsilon$ and $(y_i - x_i), (y_j - x_j)$ lie in a same coset of $\mathcal{L} \cap M$ inside $\mathcal{L}$. Our goal is to show that in fact with good probability P contains pairs $(x_i, y_i), (x_j, y_j)$ such that $f((y_i - x_i) - (y_j - x_j)) \leq \epsilon$ and $(y_i - x_i) - (y_j - x_j)$ and v lie in the same coset of $\mathcal{L} \cap M$ inside $\mathcal{L}$. We use standard argument from [AKS01, Re] about a modified distribution to prove it. We need following proposition.

Proposition 2.8 *Let $\mathcal{L} \subset \mathbb{R}^n$ be a rank n lattice, $v \in \mathcal{L}$ such that $2 \leq f(v) \leq 3$ for a nice gauge function f. Consider the convex regions $C_{-v} = B_f(-v, 2) \cap B_f(0, 2)$ and $C_v = B_f(v, 2) \cap B_f(0, 2)$. Then $C_v = C_{-v} + v$ and Vol(C_v) = Vol(C_{-v}) = $\Omega(\frac{\text{Vol}(B_f(0,2))}{2^{O(n)}})$.*

The fact that $C_v = C_{-v} + v$ follows easily and the claimed volume bound follows from the fact that $B_f(-v/2, 1/2) \subseteq C_{-v}, B_f(v/2, 1/2) \subseteq C_v$.

Note that we have picked $x_1, \ldots, x_N$ uniformly at random from $B_f(0, 2)$, where $N = 2^{c \cdot (n+k\log(1/\epsilon))} \cdot \log R$. By proposition 2.8 $x_i \in C$ with probability at least $2^{-O(n)}$. Hence by choosing c large enough we can ensure that with high probability there is $Z \subseteq P$ such that $|Z| \geq 2^{c_1(n+k\log(1/\epsilon))}$ for a constant c_1 and for all $(x_i, y_i) \in Z$, $x_i \in C$. So in the proof of the results in the Claim 4 and the Claim 5 we could have worked over the pairs in $Z \subset P$ rather than pairs in P. So we have the following Lemma.

Lemma 2.9 *P contains pairs $(x_i, y_i), (x_j, y_j)$ such that $f((y_i - x_i) - (y_j - x_j)) \leq \epsilon$ and*

$(y_i - x_i), (y_j - x_j)$ *lie in a same coset of* $\mathcal{L} \cap M$ *inside* $\mathcal{L}$ *and with probability* $1 - 2^{-\Omega(n)}$ $x_i, x_i \in C_v$.

We have $(x_i, y_i), (x_j, y_j) \in P, i \neq j, x_i, x_j \in C_{-v}$ such that $f(\beta_i - \beta_j) \leq \epsilon$ and $\beta_i - \beta_j \in \mathcal{L} \cap M$. Now, we apply the argument as explained in Regev's notes [Re] to reason with a modified distribution of the x_i. Note that in the sieving procedure described before Theorem 2.7 x_i's are chosen uniformly at random from $B_f(0, 2)$. Now we define a new distribution which is also a uniform distribution on $B_f(0, 2)$.

First lets define a bijection τ on $B_f(0, 2)$. $\tau(x) = x + v$ if $x \in C_{-v}$. $\tau(x) = x - v$ if $x \in C_v$ and $\tau(x) = x$ if $x \in B_f(0, 2) \setminus (C_{-v} \cup C_v)$. It is clear that τ is a bijection and it maps C_{-v} to C_v and C_v to C_{-v}.

Now consider the following modification in the step 1 of the algorithm. After choosing x_i with probability $\frac{1}{2}$ we replace it by $\tau(x_i)$ and with probability $\frac{1}{2}$ we keep it as it is. It is clear that this new distribution is also an uniform distribution on $B_f(0, 2)$. Since $x_i(\mod \mathcal{L}(B)) = \tau(x_i)(\mod \mathcal{L}(B))$, this modified algorithm behaves exactly in same way as the original algorithm. Now recall that we have $(x_i, y_i), (x_j, y_j) \in Z$ with $x_i, x_j \in C_{-v}$ and $f(\beta_i - \beta_j) \leq \epsilon$. Putting it together with the above argument, it follows that with probability $1 - 2^{-\Omega(n)}$ we have a tuple $(\tau(x_i), y_i) \in P$. This implies that with probability $1 - 2^{-\Omega(n)}$ we will see $v + (\beta_i - \beta_j)$ as the difference of $y_i - x_i$ and $y_j - x_j$ for some two pairs $(x_i, y_i), (x_j, y_j) \in P$. This proves the Theorem 2.7. ∎

An immediate consequence of the Theorem 2.7 we get the following Corollary.

Corollary 2.10 *Given a rank* n *lattice* $\mathcal{L}$ *and a* k*-dimensional subspace* $M \subset \mathbb{R}^n$*, there is* $1 + \epsilon$ *randomized approximation algorithm for SAP (for any nice gauge function) with running time* $2^{O(n + k \log \frac{1}{\epsilon})} \cdot poly(size(\mathcal{L}, M))$.

The $1 + \epsilon$ approximation algorithm in [BN07] for SAP has running time $2^{O(n \log \frac{1}{\epsilon})} \cdot poly(size(\mathcal{L}, M)))$. Our algorithm has running time $2^{O(n + k \log \frac{1}{\epsilon})}$ for computing $1 + \epsilon$ approximate solution. Put another way, for $k = o(n)$ we get a $2^{O(n)}$ time algorithm for obtaining $1 + 2^{-n/k}$ approximate solutions to SAP.

There is a crucial difference in our analysis of the AKS sieving and that in [BN07]. In [BN07] it is shown that with probability $1-2^{-\Omega(n)}$ the sieving procedure outputs a $1+\epsilon$ approximate solution $u \in \mathcal{L} \setminus M$.

On the other hand, we show in Claim 4 that with probability $1-2^{-\Omega(n)}$ the sieving procedure samples $2^{O(n+k\log(1/\epsilon)}$ lattice points in *some* coset of the sublattice $\mathcal{L} \cap M$ in $\mathcal{L}$. Then we argue that with probability $1-2^{-\Omega(n)}$ the sample contains a lattice point u in $\mathcal{L} \cap M + v$ such that such that $d(u,v)$ is small, for some shortest vector v in $\mathcal{L} \setminus M$. We argue this in Claim 5 by a packing argument in the coset of $\mathcal{L} \cap M$. As $\mathcal{L} \cap M$ has rank k, the packing argument in k dimensions gives the improved running time for our approximation algorithm for the problem.

The fact that the AKS sampling contains many points from the same coset of $\mathcal{L} \cap M$ also plays crucial role in our exact algorithm for SAP shown in Theorem 2.12.

2.4 Applications

The results of this section are essentially applications of ideas from Theorem 2.7 and Section 2.3.

First we describe an exact algorithm for SAP. We prove our result for full rank lattices, but it is easy to see that the result holds for general lattices as well. Let $\mathcal{L} \subset \mathbb{Q}^n$ is a full rank lattice given by a basis $\{b_1, \cdots, b_n\}$ and let $M \subseteq \mathbb{R}^n$ is a subspace of dimension $k < n$. For the ℓ_p norm, we give a randomized $2^{O(n+k\log k)}poly(s)$ time algorithm to find a shortest vector in $\mathcal{L} \setminus M$, where $s = size(\mathcal{L}, M)$. Our exact algorithm uses the same sieving procedure and the analysis described in the proof of Theorem 2.7 in Section 2.3.

As before, by considering polynomially many scalings of the lattice, we can assume that a shortest vector $v \in \mathcal{L} \setminus M$ satisfies $2 \leq \|v\|_p \leq 3$. We now describe the algorithm.

1. Apply Algorithm 1 on input lattice $\mathcal{L}$ and the subspace M of dimension k, Let ϵ be an arbitrary constant. Let $P = \{(x_i, y_i) | i \in T\}, T \subset [N]$ be the set of tuples output by the algorithm. For all $i, j \in T$ compute lattice points $v_{i,j} = (y_i - x_i) - (y_j - x_j)$.

2. Let $w_{i,j}$ is a closest lattice vector to $v_{i,j}$ in the rank k lattice $\mathcal{L} \cap M$ and let $r_{i,j} = v_{i,j} - w_{i,j}$. Output a vector of least non zero ℓ_p norm among all the vectors $r_{i,j}$ for $i, j \in T$.

First we prove the correctness of the algorithm.

Lemma 2.11 *For an appropriate choice of the constant c in the step 1 of the Algorithm 1, the above algorithm outputs a shortest non zero vector in $\mathcal{L} \setminus M$ with respect to ℓ_p norm.*

Proof Let v be a shortest vector in $\mathcal{L} \setminus M$. Since we have chosen ϵ as a constant by Theorem 2.7, it follows that there is a constant c which we can choose in step 1 of the algorithm 1 such that with probability $1 - 2^{-O(n)}$ there exists $(x_i, y_i), (x_j, y_j) \in P$ such that the lattice point $u = (x_i - y_i) - (x_j - y_j)$ and v lie in the same coset of $\mathcal{L} \cap M$ inside $\mathcal{L}$. So in the Step 2 of the the algorithm we have some $v_{i,j}$ such that $v_{i,j}$ and v lie in the same coset of $\mathcal{L} \cap M$ inside $\mathcal{L}$ i.e. $w = v_{i,j} - v \in \mathcal{L} \cap M$.

Let $w_{i,j} \in \mathcal{L} \cap M$ be a closest vector to $v_{i,j}$. So we have $\|v_{i,j} - w_{i,j}\|_p \leq \|v_{i,j} - w\| = \|v\|_p$. Since $\|v\|_p = sh(\mathcal{L}, M)$, we have $\|v_{i,j} - w_{i,j}\|_p = \|v\|_p$, which proves the lemma. ∎

Now we argue about the running time of the algorithm. In the step 1 we are invoking Algorithm 1 which runs in time $2^{O(n)}$ for a constant ϵ. Note that $\mathcal{L} \cap M$ is a rank k lattice and basis of it can be computed efficiently which follows from Lemma 2.6. In Step 2 of the algorithm we are solving $2^{O(n)}$ many instances of CVP for the rank k lattice $\mathcal{L} \cap M$. For $i, j \in S$ a closest vector to $v_{i,j}$ in the rank k lattice $\mathcal{L} \cap M$ can be computed in $2^{O(k \log k)}$ time using Kannan's algorithm for CVP [Kan87]. Hence the Step 2 takes $2^{O(n+k \log k)}$ time. Therefore the overall running time of the algorithm is $2^{O(n+k \log k)} \cdot poly(s)$. Note that by repeating above algorithm $2^{O(n)}$ times we can make the success probability of the algorithm exponentially close to 1.

Theorem 2.12 *Given a full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ and a subspace $M \subseteq \mathbb{R}^n$ of dimension $k < n$, There is a randomized algorithm to finds $v \in \mathcal{L} \setminus M$ with least possible l_p norm.*

The running time of the algorithm is $2^{O(n+k\log k)}$ times a polynomial in the input size and it succeeds with probability $1-2^{-cn}$ for an arbitrary constant c.

Given an integer lattice $\mathcal{L}$, Blömer and Naewe [BN07] gave $2^{O(n)}$ time $1+\epsilon$ factor approximation algorithm to solve SMP and SIVP. As a simple consequence of Theorem 2.12 we get a $2^{O(n)}$ time randomized algorithm to "partially" solve SMP: we can compute the first $O(\frac{n}{\log n})$ successive minima in $2^{O(n)}$ time. More precisely, we can compute a set of i linearly independent vectors $\{v_1, v_2, \ldots, v_i\} \subset \mathcal{L}$ such that $\|v_j\|_p = \lambda_j^p(\mathcal{L})$ for $j = 1$ to i if i is $O(\frac{n}{\log n})$.

Given a lattice $\mathcal{L}$, let $M \subset \mathbb{R}^n$ be a subspace of dimension zero generated by $0 \in \mathbb{R}^n$ and consider SAP instance $(\mathcal{L}, M)$. It is clear that v_1 is a shortest vector in $\mathcal{L} \setminus M$ so using Theorem 2.12 we can compute v_1 in $2^{O(n)}$ time. Now inductively assume that we have computed linearly independent vectors $v_1, v_2, \ldots, v_k \in \mathcal{L}$ such that $\|v_j\|_p = \lambda_j^p(\mathcal{L})$. Next consider $(\mathcal{L}, M)$ as instance of SAP where M is space generated by $v_1, \ldots, v_k$ and compute $v \in \mathcal{L} \setminus M$ using Theorem 2.12 in time $2^{O(n+k\log k)}$. It is clear that $\|v\|_p = \lambda_{k+1}^p(\mathcal{L})$ and as $v \notin M$ the vectors $v_1, v_2, \ldots, v_k, v$ are linearly independent. If k is $O(\frac{n}{\log n})$ it is clear that algorithm takes $2^{O(n)}$ time this proves Corollary 2.13.

Corollary 2.13 *Given a full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ and a positive integer $i \leq \frac{cn}{\log n}$ for a constant c, there is a randomized algorithm with running time $2^{O(n)} \cdot poly(size(\mathcal{L}))$ to compute linearly independent vectors $v_1, v_2, \ldots, v_i \in \mathcal{L}$ such that $\|v_j\|_p = \lambda_j^p(\mathcal{L})$ for $j = 1$ to i.*

Closest vector problem can be reduced to SAP as noted in [BN07]. Recently D. Micciancio ([Mi08]) has shown that in fact CVP, SAP and SMP are polynomially equivalent(Theorem in [Mi08]). Our algorithm computes $v \in \mathcal{L} \setminus M$ with least norm by solving $2^{O(n)}$ instances of CVP. We have basically given a randomized $2^{O(n)}$ time Turing reduction from SAP to CVP. An interesting property of our reduction is that we are solving instance $(\mathcal{L}, M)$ of SAP by solving $2^{O(n)}$ many CVP instances $(\mathcal{L} \cap M, v)$ where $\mathcal{L} \cap M$ is a rank k lattice, where k is dimension of M. Whereas for the CVP instance (N, v) produced by the SAP to CVP reduction in [BN07], the lattice N has rank $O(n)$.

As a consequence of this property of our reduction we get the Corollary 2.14 which

states that it suffices to look for a $2^{O(n)}$ randomized exact algorithm for CVP that also has access to all the successive minimas of the input lattice.

Corollary 2.14 *Suppose for all m there is a $2^{O(m)}$ randomized exact algorithm for* CVP *that takes as input a* CVP *instance (M, v) where M is full rank lattice of rank m and $v \in \mathbb{R}^m$ (along with the* extra *input $v_i \in M$ such that $|v_i|_p = \lambda_i^p(M)$ for $i = 1$ to m where $\lambda_i^p(M)$ is i^{th} successive minima in M). Then, in fact, there is a $2^{O(n)}$ randomized exact algorithm for solving* CVP *on any rank n lattice.*

Proof By [Mi08] CVP is poly-time equivalent to successive minima problem(SMP). Consider full rank lattice $\mathcal{L} \subset \mathbb{Q}^n$ as input to SMP. So it suffices to compute linearly independent $v_1, \ldots, v_n \in \mathcal{L}$ with $\|v_i\|_p = \lambda_i^p(\mathcal{L})$ for $i = 1$ to n in $2^{O(n)}$ time. We proceed as in the proof of Corollary 2.13. Inductively assume that we have already computed linearly independent vectors $v_1, \ldots, v_k \in \mathcal{L}$ with $\|v_i\|_p = \lambda_i^p(\mathcal{L})$. Let M is space generated by $v_1, \ldots, v_k$. As in proof of Theorem 2.12 we can solve $(\mathcal{L}, M)$ an instance of SAP by solving $2^{O(n)}$ many instances of CVP $(\mathcal{L} \cap M, v')$. Note that $\mathcal{L} \cap M$ is rank k lattice and it is clear that $\|v_i\|_p \lambda_i^p(\mathcal{L} \cap M)$ for $i = 1$ to k. So we can solve these instances in $2^{O(n)}$ time.(Note that though $\mathcal{L} \cap M$ is not full rank lattice, but it is not difficult to convert all these instances of CVP to full rank by applying suitable orthonormal linear transformation.) This takes time $2^{O(n+k)}$ which is at most $2^{O(n)}$. So it is clear that we can compute linearly independent vectors $v_1, \ldots, v_n \in \mathcal{L}$ with $\|v_i\|_p = \lambda_i^p(\mathcal{L})$ in time $n \cdot 2^{O(n)}$. This proves the Corollary. ∎

In the next corollary we give a $2^{O(n)}$ time algorithm to solve certain CVP instances $(\mathcal{L}, v)$ for any ℓ_p norm. Let $\lambda_i^p(\mathcal{L})$ denotes i th successive minima of the lattice $\mathcal{L}$ with respect to ℓ_p norm.

Corollary 2.15 *Let $(\mathcal{L}, v)$ be a* CVP *instance such that $\mathcal{L}$ is full rank with the promise that*

$$d(v, \mathcal{L}) < \frac{(2^p - 1)^{\frac{1}{p}}}{2} \lambda_t^p(\mathcal{L})$$

where $t \leq \frac{cn}{\log n}$ and c is an absolute constant. Then there is a $2^{O(n)} \cdot poly(size(\mathcal{L}))$ time randomized algorithm that solves the CVP *instance $(\mathcal{L}, v)$ exactly.*

Proof . By Corollary 2.13 we first compute $\lambda_t^p(\mathcal{L})$. We now use ideas from Kannan's CVP to SVP reduction [Kan87]. Let $b_1, b_2, \cdots, b_n$ be a basis for $\mathcal{L}$. We obtain new vectors $c_i \in \mathbb{Q}^{n+1}$ for $i = 1$ to n by letting $c_i^T = (b_i^T, 0)$. Likewise, define $u \in \mathbb{Q}^{n+1}$ as $u^T = (v^T, \lambda_t^p(\mathcal{L})/2)$. Let $\mathcal{M}$ be the lattice generated by the $n+1$ vectors $u, c_1, c_2, \cdots c_n$. Compute the vectors $v_j \in \mathcal{M}$ such that $\|v_j\|_p = \lambda_j^p(\mathcal{M})$ for $j = 1$ to t using Corollary 2.13 in time $2^{O(n)} \cdot poly(size(\mathcal{L}))$. Write vectors v_j as $v_j = u_j + \alpha_j u$, $u_j \in \mathcal{L}(c_1, \cdots, c_n)$ and $\alpha_j \in \mathbb{Z}$. Since u has $\lambda_t^p(\mathcal{L})/2$ as its $(n+1)^{th}$ entry, $|\alpha_j| \leq 1$ for $1 \leq j \leq t$. If $|\alpha_j| > 1$ we have $|\alpha_j| \geq 2$ and which implies $\|v_j\|_p \geq \lambda_t^p(\mathcal{L}) > \lambda_t^p(\mathcal{M})$, a contradiction so we must have $|\alpha_j| \leq 1$ for $1 \leq j \leq t$. As

$$d(v, \mathcal{L}) < \frac{(2^p - 1)^{\frac{1}{p}}}{2} \lambda_t^p(\mathcal{L})$$

we have

$$d(u, \mathcal{L}(c_1, \cdots, c_n))^p = d(v, \mathcal{L})^p + (\frac{\lambda_t^p(\mathcal{L})}{2})^p < \lambda_t^p(\mathcal{L})^p$$

. Hence, there is at least one index i, $1 \leq i \leq t$ such that $|\alpha_i| = 1$. Consider the set $S = \{u_i \mid 1 \leq i \leq t, |\alpha_i| = 1\}$and let u_j be the shortest vector in S. Writing $u_j^T = (w_j^T, 0)$, it is clear that the vector $-w_j \in \mathcal{L}$ is closest vector to v if $\alpha_j = 1$ and w_j is a closest vector to v if $\alpha_j = -1$. ∎

2.5 Overview

In this chapter we explored the algorithmic complexity of various lattice problems like Shortest Vector Problem(SVP), Closest Vector Problem(CVP), Successive Minima Problem(SMP), Subspace Avoiding Problem(SAP) based on sieving technique developed by Ajtai, Kumar and Sivakumar [AKS01]. We know that the exact solution for SVP can be found in randomized $2^{O(n)}$ time with respect to any ℓ_p norm [AKS01], whereas the best known exact algorithms for CVP, SMP, SAP with respect to general ℓ_p norm have running time $2^{O(n \log n)}$ ([Kan87], [Mi08]). These problems also have a $2^{O(n)}$ randomized $1+\epsilon$ approximation algorithm with respect to general ℓ_p norm for a constant $\epsilon > 0$ ([AKS02], [BN07]). In a special case of ℓ_2 norm [MV10] gives a deterministic $2^{O(n)}$ time algorithm for CVP, SAP and SMP. The question of getting exact solution of CVP, SAP, SMP in $2^{O(n)}$ time remains open for general ℓ_p norm.

In the work presented in this chapter we made an effort to understand the complexity of finding an exact solution for SAP and CVP with respect to general ℓ_p norms. First we give a new parameterized algorithm to solve SAP, running time of which is sensitive to the dimension of the subspace involved. Given a rank n integer lattice $\mathcal{L}$ and a subspace $M \subset \mathbb{R}^n$ of dimension k our algorithm runs in time $2^{O(n+k\log k)}$ and gives an exact solution for SAP with respect to ℓ_p norm. In particular if the dimension of the subspace is $O(\frac{n}{\log n})$ then we get a $2^{O(n)}$ algorithm to solve SAP exactly with respect to any ℓ_p norm. This algorithm is based on the AKS-sieving. [BN07] also give algorithm for SAP based on the AKS-sieving. Our algorithm performs better, parameterized on the dimension of the subspace because in our analysis we exploit the coset structure of the lattice $\mathcal{L} \cap M$ inside $\mathcal{L}$. This enable us to sample lattice points from a coset of a shortest vector in $\mathcal{L} \setminus M$ and apply packing argument within the coset. An obvious implication of this algorithm is, we can compute first $O(\frac{n}{\log n})$ successive minimas of the lattice with respect to ℓ_p norm in $2^{O(n)}$ time. We also get a new $1+\epsilon$ approximation algorithm for SAP with respect to any ℓ_p norm and runs in time $2^{O(n+k\log(1/\epsilon))}$ which is improvement over the algorithm for SAP in [BN07], parameterized on the dimension of the subspace.

As an implication of our coset sampling technique and the parameterized algorithm for SAP we obtained some new results for Closest Vector Problem. We gave a $2^{O(n)}$ time algorithm for CVP if the input vector satisfies certain promise about the distance from the input lattice. We showed that to get a $2^{(O(n)}$ time algorithm for CVP with respect to ℓ_p norm, it is enough to get such an algorithm assuming that the information about the successive minimas of the input lattice is given for free.

From the results in this chapter it follows that the AKS-sieving not only works for all ℓ_p norms, but also for a more general notion of norm specified by a *gauge function* [Si45]. We need a mild restriction on the skewness of the convex body associated with the gauge function to enable a randomized sampling from the convex-body using [DFK91]. We call the gauge function satisfying this skewness restriction as *nice gauge functions* (which includes all ℓ_p norms) and AKS-sieving works for all nice gauge functions in fact we need only oracle access to the gauge function. An interesting question in this context is, can we prove a lower bound on the number of queries made by the algorithm for CVP and SVP which accesses a gauge function given as a black box oracle?

Another important issue is the space used by the algorithms for the lattice problems .

Sieving algorithms for SVP, CVP, SAP all use an exponential amount of space. The recent algorithm for CVP [MV10] which is not based on the sieving procedure also uses an exponential amount of the space. Whereas Kannan's algorithm for CVP has slightly super-exponential running time ($2^{O(n \log n)}$) but it uses only a polynomial amount of space and works with respect to general ℓ_p norms. Important open problem in the area is getting an exponential time algorithm for CVP with respect to ℓ_p norms which uses only polynomial amount of space.

3

Algorithmic Problems for Metrics on Permutation Groups

In this chapter we investigate the computational complexity of Minimum Weight Problem(MWP) and Subgroup Distance Problem(SDP) for metrics on permutation groups. These problems are analogous to shortest vector problem (SVP) and closest vector problem(CVP) for integer lattices considered in the previous chapter. Our motivation to study these problems comes from the fact that MWP and SDP are appropriate analogues of SVP and CVP in the *noncommutative* setting. Our aim is to investigate whether some of the algorithmic results for CVP and SVP for integer lattices (which are abelian groups) can be extended to the analogous results in the noncommutative setting. It turns out that in some of the cases, tools and techniques used for lattice problems can be extended to study MWP and SDP over permutation groups. In other cases we apply techniques from permutation group theory to study MWP and SDP.

3.1 Introduction

Motivated by the generic nature of the AKS-sieving procedure for Shortest Vector Problem(SVP) studied in the last chapter, it is natural to ask whether it can work for similar optimization problems in the other domains. Specifically, we investigate the computational complexity of the two natural problems for metrics on permutation groups given

by generating sets. These problems are *noncommutative* analogues of SVP and CVP. We are also interested in comparing the techniques used to study these problems with the techniques used to study SVP and CVP.

Several permutation metrics are studied in the literature and they have applications in areas such as statistics, coding theory, computing. We refer to [DH98] for a survey on metrics on permutation groups and pointers to various applications.

Metrics on permutations are used to address several statistical problems associated with *partially ranked data*. Such problems in their simplest form arise in situations such as the following: Suppose there are t persons and a set of n items. Each person ranks the first k items of his choice for $k < n$. We need to address statistical questions that involve measuring the degree of association between the partial rankings of any two persons. Or we need to figure out if this partial ranking data point to a significant statistical difference between two different sub-populations of rankers etc. Critchlow, in [Cr85] describes several applications of metrics on permutation groups to such problems about partially ranked data. In a different line of work, Blake et al, in [BCD79], explore whether permutation groups can be used as an error correcting code with respect to some general metric on permutation. This question leads to several interesting problems about permutation groups. Another area in which metrics on permutation arise is extremal permutation group theory. Various extremal problems over permutation groups are explored. These are analogues of extremal problems in set theory. For work in this direction and pointers to interesting results we refer to [Ca88], [DF77].

Preliminaries

We will introduce some notation and give formal definitions. We first recall some permutation group theory relevant for this chapter.

Let S_n denote the group of all permutations on a set Ω of size n (usually, $\Omega = \{1, \cdots, n\}$). In general, we refer to any *subgroup* G of S_n as a permutation group. A subset S of a group G is a *generating set* for G if the smallest subgroup of G containing S is G. Every element of G can be expressed a product of elements of a generating set S. As a consequence of Lagrange's theorem, every finite group G has a generating set of size

$\log_2 |G|$. Thus, every subgroup of S_n has a generating set of size $O(n \log n)$.

Let G be a subgroup contained in S_n. For each $i \in [n]$, G has the subgroup $G_i = \{g \in G | g(i) = i\}$ consisting of all elements of G that fix i. Likewise, for any subset $I \subseteq [n]$ we define the subgroup that *pointwise stabilizes* I:

$$G_I = \{g \in G | g(j) = j \text{ for each } j \in I\}.$$

If we denote $G^{(i)} = G_{[i-1]}$ for $1 \leq i \leq n$ then we have a chain of stabilizers

$$G = G^{(0)} \geq G^{(1)} \geq \ldots G^{(n)} = e$$

with the property that $\frac{|G^{(i)}|}{|G^{(i+1)}|} \leq n - i$.

By Lagrange's theorem $G^{(i)}$ is a disjoint union of at most $n - i$ right cosets of $G^{(i+1)}$. More precisely, we can write

$$G^{(i)} = G^{(i+1)}\sigma_1 \cup \cdots \cup G^{(i+1)}\sigma_k,$$

for some $k \leq n - i$. These permutations σ_j are called *coset representatives*.

Definition 3.1 *A strong generating set for G is a set S of permutations such that for each $i = 1, 2, \ldots, n-1$, $(S \cap G^{(i)}) \setminus G^{(i+1)}$ is a complete set of distinct coset representatives for $G^{(i)}$.*

Given a generating set for a permutation group G, the Schreier-Sims algorithm enable us to compute a *strong generating set* of $O(n^2)$ size for G in deterministic polynomial time. We refer to [Lu93] for the details of the algorithm and its several applications. As a consequence of the Schreier-Sims algorithm we can devise polynomial-time algorithms to do following basic tasks.

- Computing order of a permutation group.
- Testing membership in a given permutation group.
- Uniformly sampling an element from a given permutation group.

- Given a permutation group G acting on Ω, computing orbit of any point $k \in \Omega$. Orbit of point k is $\{i \in \Omega |$ there exist $g \in G s.t. g(k) = i\}$.

Metrics on Permutations

A function $d : S_n \times S_n \mapsto \mathbb{R}$ is a *metric* on the permutation group S_n if:

- For all $\pi, \tau, \psi \in S_n$, $d(\pi, \tau) = d(\tau, \pi) \geq 0$ and $d(\pi, \tau) = 0$ if and only if $\pi = \tau$.
- Furthermore, the triangle inequality holds: $d(\pi, \tau) \leq d(\pi, \psi) + d(\psi, \tau)$.

Let $e \in S_n$ denote the identity permutation of S_n. For $\tau \in S_n$, $d(e, \tau)$ is the *norm* of τ for metric d, and is denoted by $\|\tau\|$.

A *right-invariant metric* d on S_n satisfies $d(\pi, \tau) = d(\pi\psi, \tau\psi)$ for all $\pi, \tau, \psi \in S_n$. A *left-invariant metric* is similarly defined. We say d is bi-invariant if it is both right and left invariant. A detailed discussion about metrics on permutations can be found in [DH98]. We recall the definitions of some permutation metrics useful for this chapter.

Hamming distance: $d(\tau, \pi) = |\{i | \tau(i) \neq \pi(i)\}|$.

l_p **distance ($p \geq 1$):** $d(\tau, \pi) = (\Sigma_{i=1}^{n} |\tau(i) - \pi(i)|^p)^{1/p}$.

l_∞ **distance:** $d(\tau, \pi) = max_{1 \leq i \leq n} |\tau(i) - \pi(i)|$.

Cayley Distance: $d(\tau, \pi) =$ minimum number of transpositions taking τ to π.

These metrics are all right invariant. The Hamming and Cayley metrics are also left invariant.

For $S \subseteq S_n$ and $\tau \in S_n$ let

$$d(\tau, S) = min_{\psi \in S} d(\tau, \psi).$$

For $\tau \in S_n, r \in \mathbb{R}^+$ let

$$B_n(\tau, r, d) = \{\pi \in S_n | d(\pi, \tau) \leq r\}$$

be the ball of radius r centered at τ for a metric d. Analogous to the geometric setting, we define the volume $\text{Vol}(S)$ of a subset $S \subseteq S_n$ as its size $|S|$. For right invariant metric d we have, for all $\tau \in S_n, r \geq 0$, $\text{Vol}(B_n(e, r, d)) = \text{Vol}(B_n(\tau, r, d))$.

We now define the Subgroup Distance Problem and the Minimum Weight Problem with respect to a metric d.

Definition 3.2 [CW06, BCW06]
Subgroup Distance Problem (SDP): *Input instances are (G, τ, k), where $G \leq S_n$ is given by a generating set, $\tau \in S_n$, and $k > 0$. Is $d(\tau, G) \leq k$?*

Minimum Weight Problem (MWP): *Input instances are (G, k), $G \leq S_n$ given by a generating set and $k > 0$. Is there a $\tau \in G \setminus \{e\}$ with $\|\tau\| \leq k$?*

We are also interested in approximate solutions to MWP and SDP. For MWP, given $\gamma > 1$ the problem is to find a $\pi \in G$, $\pi \neq e$ such that $\|\pi\|$ is bounded by γ times the optimal value. Likewise for SDP. As usual, we can define promise decision versions of SDP and MWP that capture this notion of approximation.

For any permutation metric d, the promise problem GapSDP_γ where γ is a function of n, is defined as follows: inputs are the SDP inputs (G, τ, k). An instance (G, τ, k) is a YES instance if there exist $\psi \in G$ such that $d(\psi, \tau) \leq k$. And (G, τ, k) is a NO instance if for all $\psi \in G$, $d(\psi, \tau) > \gamma k$. The problem GapMWP_γ is similarly defined. An algorithm *solves* the promise problem if it decides correctly on the YES and NO instances.

Note that if we can compute γ-approximate solution for MWP (SDP resp.) we can solve corresponding promise problem GapMWP_γ (GapSDP_γ resp.). To see this suppose we can compute γ-approximate solution for MWP, i.e. we can compute $\tau \in G$ such that $\|\tau\| \leq \gamma t$ where t is norm of shortest non-identity permutation in G. To solve an instance (G, k) of GapMWP_γ we simply check if $\|\tau\| > \gamma k$, if so then we have $\gamma t > \gamma k$ so $t > k$. This implies (G, k) is not a YES instance of GapMWP_γ İn other case when $\|\tau\| \leq \gamma k$, which implies (G, k) is not a NO instance of GapMWP_γ Ṡimilarly if we can compute γ approximate solution for SDP we can solve the promise version of SDP.

Results in this chapter

We study the complexity of MWP with respect to Hamming and ℓ_∞ metrics. Hamming distance between permutations $\tau, \pi \in S_n$ is defined as $d(\tau, \pi) = |\{i|\tau(i) \neq \pi(i)\}|$. ℓ_∞ distance between τ, π is $d(\tau, \pi) = max_{1 \leq i \leq n}|\tau(i) - \pi(i)|$. MWP is NP-hard with respect to both of these metrics even for abelian permutation groups [CW06].

A naive brute-force search algorithm for MWP (which enumerates all the permutations and finds a permutation in G with shortest nonzero norm) can take upto $n!$ steps since $G \leq S_n$ can have up to $n!$ elements. It easily follows that if $G \leq S_n$ is an abelian group then $|G| \leq 2^{O(n)}$, so using classical Schrier-Sims algorithm we can enumerate all the permutations in G and find one with the smallest nonzero norm. This gives a $2^{O(n)}$ algorithm to solve MWP for abelian groups. More interesting case is that of the nonabelian permutation groups.

In the case of Hamming metric we give a deterministic $2^{O(n)}$ time algorithm which is *group theoretic* in nature. The algorithm is based on the classical Schrier-Sims algorithm. However, the problem for l_∞ metric does not appear amenable to a permutation group-theoretic approach. We give a $2^{O(n)}$ time randomized algorithm for the problem. Interestingly, for this algorithm we are able to adapt ideas from the Ajtai-Kumar-Sivakumar algorithm for the shortest vector problem for integer lattices [AKS01]. As seen in Chapter 2, basic idea of AKS algorithm is first to pick a large number of lattice points randomly and perturb them with a certain distribution. Then apply the *sieving procedure* on these perturbed lattice points successively to get shorter and shorter lattice points. Our algorithm uses similar procedure, but since the nice geometrical structure is missing while dealing with the case of permutation groups, sampling and perturbation procedure is bit more complex.

Other results presented in this chapter include,

- It is known that SDP is NP-hard([BCW06]) and it easily follows that SDP is hard to approximate within a factor of $\log^{O(1)} n$ unless P=NP. In contrast, we show that SDP for approximation factor more than $n/\log n$ is unlikely to be NP-hard unless coNP has constant round interactive proof system, with a constant error probability and verifier is allowed to use $n^{O(\log n)}$ running time. Such a protocol

for problems in coNP is believed to be unlikely. This result adapts ideas from similar result in case of integer lattices [GG00].

- In case of integer lattices as well as for binary linear codes it is known that CVP reduces to SVP under polynomial-time reduction [GMSS99], where as getting such a reduction from SVP to CVP is an open problem. For several permutation metrics, we show that the minimum weight problem is polynomial-time reducible to the subgroup distance problem for *solvable* permutation groups. Our results adapts ideas from [GMSS99].

3.2 A $2^{O(n)}$ algorithm for MWP over l_∞ metric

We consider the search version of MWP: given $G \leq S_n$, the goal is to find a permutation $\tau \in G \setminus \{e\}$ with minimum norm with respect to a metric d. We refer to such a $\tau \in G$ as a *shortest* permutation in G w.r.t. the metric d. [CW06] shows that a decision version of MWP is NP-hard for various metrics including ℓ_∞, Cayley, Hamming metrics.

First we consider the l_∞ metric and give a $2^{O(n)}$ time randomized algorithm for finding a shortest permutation for $G \leq S_n$ given by generating set. The algorithm uses the framework developed in [AKS01] for the shortest vector problem for integer lattices. Regev's notes [Re] contains a nice exposition.

The basic idea is to first pick N elements of G independently and uniformly at random, where N is $2^{c \cdot n}$ (where the constant c will be appropriately chosen). Each of these elements is multiplied by a random permutation of relatively smaller norm to get a new set of N elements. On this set of permutations a sieving procedure is applied in several rounds. The crucial property of the sieving is that after each stage the remaining permutations have the property that the maximum norm is halved and in the process at most $2^{c' \cdot n}$ elements are sieved out for a small constant c'.

Thus, repeated sieving reduces the maximum norm until it becomes a constant multiple of norm of shortest permutation of G. Then we can argue that for some π_1, π_2 from the final set of permutations, $\pi_1 \pi_2^{-1}$ will be a shortest permutation with high probability.

First we prove certain volume bound for l_∞ metric ball, which is crucially used in the

algorithm, next we give a procedure to sample permutations from an l_∞ metric ball uniformly.

Lemma 3.3 *For $1 \leq r \leq n-1$ we have, $r^n/e^{2n} \leq$ Vol$(B_n(e,r,l_\infty)) \leq (2r+1)^n$. Consequently, for any constant $\alpha < 1$, Vol$(B_n(e,r,l_\infty))/$Vol$(B_n(e,\alpha r,l_\infty)) \leq 2^{c_1(\alpha)\cdot n}$, where c_1 is a constant which depends on the choice of α.*

Proof Let $\tau \in B_n(e,r,l_\infty)$. So, $|\tau(i)-i| \leq r$ for all i. Thus, for each i there are at most $2r+1$ choices for $\tau(i)$. This implies Vol$(B_n(e,r,l_\infty)) \leq (2r+1)^n$. Although better bounds can be shown, this simple bound suffices for the lemma. Now we show the claimed lower bound. Let $n = kr+t$, $0 \leq t \leq r-1$. For $jr+1 \leq i \leq (j+1)r, 0 \leq j \leq k-1$, $\tau(i)$ can take any value in $\{jr+1, jr+2, \ldots, (j+1)r\}$. Hence we have Vol$(B_n(e,r,l_\infty)) \geq r!^k t! \geq (r^r/e^r)^k t! \geq r^{n-t}t^t/e^n$. Using some calculus it is easily seen that the function $y = r^{n-t}t^t$ is minimum at $t = r/e$. Hence $r^{n-t}t^t/e^n \geq r^n/e^{n+r/e} \geq r^n/e^{2n}$. This proves the first part of lemma. The second part is immediate. ∎

We now explain a uniform random sampling procedure from $B_n(e,r,l_\infty)$. First, we randomly generate a function $\tau \in [n]^{[n]}$ by successively assigning values to $\tau(i)$ for $i \in [n]$ as follows. For each $i \in [n]$ we have the list $L_i = \{j | 1 \leq j \leq n, i-r \leq j \leq i+r\}$ of candidate values for $\tau(i)$. Thus we have at most $(2r+1)^n$ functions from which we uniformly sample τ. Of course, τ defined this way need not be a permutation, but if it is a permutation then clearly $\tau \in B_n(e,r,l_\infty)$. Our sampling procedure outputs τ if it is a permutation and outputs "fail" otherwise. By Lemma 3.3 the probability that τ is a permutation is

$$\text{Prob}[\tau \in B_n(e,r,l_\infty)] \geq \frac{r^n}{e^{2n}(2r+1)^n} > \frac{1}{24^n} > 2^{-5n}.$$

Thus, if we repeat above procedure sufficiently many times say 2^{10n} times then the probability that the sampling procedure fails all the times is at most $(1-2^{-5n})^{2^{10n}}$ so the failure probability is bounded by $e^{-2^{5n}}$. So with probability $1-e^{-2^{5n}}$ procedure succeeds and when it succeeds it uniformly samples from $B_n(e,r,l_\infty)$. In summary we have the following lemma.

Lemma 3.4 *There exists a randomized procedure which runs in time $2^{O(n)}$ and produces a uniform random sample from $B_n(e, r, l_\infty)$ and the procedure succeeds with probability at least $1 - e^{-2^{5n}}$.*

Now we describe the sieving procedure used in the algorithm. Hereafter we denote $B_n(\psi, r, l_\infty)$ by $B_n(\psi, r)$ for simplicity.

Lemma 3.5 *[Sieving Procedure] Let $r > 0$ and $\{\tau_1, \tau_2, \tau_3, \ldots, \tau_N\} \subseteq B_n(e, r)$ be a subset of permutations. Then in $N^{O(1)}$ time we can find $S \subset [N]$ of size at most $2^{c_1 n}$ for a constant c_1 such that for each $i \in [N]$ there is a $j \in S$ with $l_\infty(\tau_i, \tau_j) \leq r/2$.*

Proof We construct S using a greedy algorithm. Start with $S = \emptyset$ and run the following step for all elements $\tau_i, 1 \leq i \leq N$. At the i^{th} step we consider τ_i. If $l_\infty(\tau_i, \tau_j) > r/2$ for all $j \in S$ include i in set S and increment i. After completion, for all $i \in [N]$ there is a $j \in S$ such that $l_\infty(\tau_i, \tau_j) \leq r/2$. To argue that $|S| < 2^{c_1 n}$ for constant c_1 we use the volume bound of Lemma 3.3. The construction of S implies for distinct indices $j, k \in S$ that $l_\infty(\tau_j, \tau_k) > r/2$. Hence the metric balls $B_n(\tau_j, r/4)$ for $j \in S$ are all pairwise disjoint. The right invariance of l_∞ metric implies $\text{Vol}(B_n(\tau_j, r/4)) = \text{Vol}(B_n(e, r/4))$. As $\tau_j \in B_n(e, r)$, by triangle inequality we have $B_n(\tau_j, r/4) \subseteq B_n(e, r + r/4)$ for $j \in S$. Hence by Lemma 3.3

$$|S| < \frac{\text{Vol}(B_n(e, 5r/4))}{\text{Vol}(B_n(e, r/4))} \leq 2^{c_1 n}$$

for a constant c_1. This completes the proof of the lemma. ∎

Now we describe our algorithm to find a shortest permutation in G using Lemma 3.5. Let t denote the norm of a shortest permutation in G. The following claim gives an easy $2^{O(n)}$ time algorithm when $t \geq n/10$.

Lemma 3.6 *If the norm t of a shortest permutation in G is greater than $n/10$ then in time $2^{O(n)}$ we can find a shortest permutation in G.*

Proof Consider $B_n(\tau, t/2)$ for $\tau \in G$. By triangle inequality, all $B_n(\tau, t/2)$ are disjoint. Also, by Lemma 3.3 we have $\text{Vol}(B_n(\tau, t/2)) \geq (t/2)^n \cdot e^{-2n} \geq n^n \beta^{-n}$ for some

constant $\beta > 1$. Since $|G| \leq |S_n|/\text{Vol}(B_n(\tau, t/2))$, it follows that $|G| \leq \beta^n$. As we can do a brute-force enumeration of G in time polynomial in $|G|$, we can find a shortest permutation in $2^{O(n)}$ time. ∎

Now we consider the case when $t < n/10$. We run the algorithm below for $1 \leq t < n/10$ (the possible values of t) and output a shortest permutation in G produced by the algorithm.

Algorithm 1

1. Let $N = 2^{cn}$. For $1 \leq i \leq N$, pick ρ_i independently and uniformly at random from G, and pick τ_i uniformly at random from $B_n(e, 2t)$.
2. Let $\psi_i = \tau_i \rho_i$, $1 \leq i \leq N$. Let $Z = \{(\psi_1, \tau_1), (\psi_2, \tau_2), \ldots, (\psi_N, \tau_N)\}$, and let $R = max_i \|\psi_i\|$.
3. Set $T = [N]$.
4. While $R > 6 * t$ do the following steps:
   (a) Apply the "sieving procedure" of Lemma 3.5 to $\{\psi_i \mid i \in T\}$. Let set $S \subseteq T$ be the output of sieving procedure.
   (b) for all $i \in S$ remove tuple (ψ_i, τ_i) from Z.
   (c) for all $i \notin S$ replace tuple $(\psi_i, \tau_i) \in Z$ by $(\psi_i \psi_j^{-1} \tau_j, \tau_i)$, where $j \in S$ and $\ell_\infty(\psi_j, \psi_i) \leq R/2$.
   (d) set $R = R/2 + 2t$.
   (e) $T := T \setminus S$.
5. For all $(\varphi_i, \tau_i), (\varphi_j, \tau_j) \in Z$, let $\varphi_{i,j} = (\tau_j^{-1} \varphi_j)(\tau_i^{-1} \varphi_i)^{-1}$ (which is in G). Output a $\varphi_{i,j}$ with smallest nonzero ℓ_∞ norm.

In the Step 1 of the Algorithm 1, a uniform random sampling procedure from l_∞ metric ball is given by Lemma 3.4. For $G \leq S_n$, uniform sampling from G can be done in polynomial time by using a strong generating set for G (see e.g. [Lu93]). A random element is obtained by picking a coset representative at each level from the point-wise

stabilizer tower of subgroups and multiplying them out. Thus Step 1 of the Algorithm 1 takes $2^{O(n)}$ time. Clearly, the while loop takes $2^{O(n)}$ time.

In order to prove that the permutation given by the Step 5 of the Algorithm 1 is a shortest permutation in G with very high probability, we first examine the invariant maintained during each iteration of the while loop.

Proposition 3.7 *Before each iteration of the while loop, the following invariant is maintained. For all $i \in T$ we have $(\varphi_i, \tau_i) \in Z$, $\tau_i^{-1}\varphi_i \in G$ and $\|\varphi_i\| \leq R$.*

Proof Clearly, the invariant holds before the first iteration. Inductively, suppose that at the beginning of an arbitrary iteration the set Z is of the form $Z = \{(\varphi_i, \tau_i) \mid i \in T\}$ such that $\tau_i^{-1}\varphi_i \in G$ and $\|\varphi_i\| \leq R$. During this iteration, in Z we replace (φ_i, τ_i) by $(\varphi_i\varphi_j^{-1}\tau_j, \tau_i)$, where $j \in S$ and $l_\infty(\varphi_i, \varphi_j) \leq R/2$. By right invariance of the l_∞ metric, we have $l_\infty(\varphi_i, \varphi_j) = \|\varphi_i\varphi_j^{-1}\| \leq R/2$. Triangle inequality implies $\|\varphi_i\varphi_j^{-1}\tau_j\| \leq \|\tau_j^{-1}\| + \|\varphi_i\varphi_j^{-1}\| = \|\tau_j\| + \|\varphi_i\varphi_j^{-1}\| \leq 2t + R/2$ which equals the value of R set in Step 4(d). Hence, $\|\varphi_i\| \leq R$ at the beginning of next iteration. Clearly, $\tau_i^{-1}\varphi_i\varphi_j^{-1}\tau_j$ is in G since $\tau_i^{-1}\varphi_i$ and $\tau_j^{-1}\varphi_j$ are in G. ∎

By Proposition 3.7 when the algorithm stops (after Step 5) we have $\tau_i^{-1}\varphi_i \in G$ and $\|\tau_i^{-1}\varphi_i\| \leq 8t$ for all $(\varphi_i, \tau_i) \in Z$. We want to argue that one of th $\varphi_{i,j}$ is equal to a shortest permutation in G with high probability. In Step 1 of the Algorithm 1 we pick τ_i uniformly at random from $B_n(e, 2t)$ using sampling procedure of Lemma 3.4. As in the Chapter 2 (or in the Regev's analysis of the AKS algorithm in [Re]), we define a new random procedure which also uniformly samples from $B_n(e, 2t)$ and has some properties which enable us to conveniently argue the correctness of the algorithm. In the lattice setting, the Euclidean metric makes it easier to define a modified sampling. However, for the l_∞ metric over S_n, the modified sampling from $B_n(e, 2t)$ is more involved.

Let $\tau \in G$ be an element with shortest nonzero norm t. We introduce some notation. Let $C_\tau = B_n(e, 2t) \cap B_n(\tau, 2t)$, $C_{\tau^{-1}} = B_n(e, 2t) \cap B_n(\tau^{-1}, 2t)$ and $C = C_\tau \cap C_{\tau^{-1}}$. We need the following Lemma.

Lemma 3.8 *Consider a map $\phi_1 : C_\tau \longrightarrow C_{\tau^{-1}}$ defined as $\phi_1(\sigma) = \sigma\tau^{-1}$. Then ϕ_1 is a bijection from C_τ onto $C_{\tau^{-1}}$.*

Proof Let $\sigma \in C_\tau$. Then $l_\infty(\sigma, e) \leq 2t$ and $l_\infty(\sigma, \tau) \leq 2t$. By right invariance it follows that, $l_\infty(\phi_1(\sigma), e) = {}_\infty(\sigma\tau^{-1}, e) = l_\infty(\sigma, \tau) \leq 2t$ and $\ell_\infty(\phi_1(\sigma), \tau^{-1}) = \ell_\infty(\sigma\tau^{-1}, \tau^{-1}) = \ell_\infty(\sigma, e) \leq 2t$. Hence $\phi_1(\sigma) \in C_{\tau^{-1}}$, so ϕ_1 is well defined. By definition ϕ_1 is one-one. Now consider $\sigma \in C_{\tau_{-1}}$. It is easy to see that $\sigma\tau \in C_\tau$ and $\phi_1(\sigma\tau) = \sigma$, So ϕ_1 is onto. This proves the desired. ∎

Let $\phi_1' : C_{\tau^{-1}} \longrightarrow C_\tau$ denote the inverse of ϕ_1.

We now define a randomized procedure Sample which on input a random permutation $\sigma \in B_n(e, 2t)$ returns a new random permutation Sample$(\sigma) \in B_n(e, 2t)$. This sampling procedure is introduced only for the purpose of the analysis and we do not need to implement it in the algorithm.

(i) If $\sigma \notin C_\tau \cup C_{\tau^{-1}}$ then Sample$(\sigma) = \sigma$ with probability 1.

(ii) If $\sigma \in C_\tau \setminus C$ then

  (a) if $\phi_1(\sigma) \in C$ then randomly set Sample(σ) to either σ with probability $3/4$ or to $\phi_1(\sigma)$ with probability $1/4$.

  (b) if $\phi_1(\sigma) \notin C$ then randomly set Sample(σ) to σ or $\phi_1(\sigma)$ with probability $1/2$ each.

(iii) If $\sigma \in C_{\tau^{-1}} \setminus C$ then define Sample(σ) analogously as in Step (ii) above, using ϕ_1' instead of ϕ_1.

(iv) If $\sigma \in C$, then randomly set Sample(σ) to either σ with probability $1/2$, or to $\phi_1(\sigma)$ with probability $1/4$, or to $\phi_1'(\sigma)$ with probability $1/4$.

Lemma 3.9 *If σ is uniformly distributed in $B_n(e, 2t)$ then* Sample(σ) *is also uniformly distributed in $B_n(e, 2t)$.*

Proof Let $V = \text{Vol}(B_n(e, 2t))$. For each $\pi \in B_n(e, 2t)$ we have

$$\text{Prob}[\text{Sample}(\sigma) = \pi] \ = \sum_{\delta \in B_n(e,2t)} \text{Prob}[\sigma = \delta] \cdot \text{Prob}[\text{Sample}(\delta) = \pi].$$

We need to show that

$$\sum_{\delta \in B_n(e,2t)} \text{Prob}[\sigma = \delta] \cdot \text{Prob}[\text{Sample}(\delta) = \pi] = 1/V.$$

As σ is uniformly distributed, it is equivalent to showing $\sum_{\delta \in B_n(e,2t)} \text{Prob}[\text{Sample}(\delta) = \pi] = 1$. If $\pi \notin C_\tau \cup C_{\tau^{-1}}$ it is true directly from the definition of Sample.

Consider $\pi \in C$, since maps ϕ_1 and ϕ_1' are bijective, there are unique σ_1, σ_2 such that $\phi_1(\sigma_1) = \phi_1'(\sigma_2) = \pi$. By definition of Sample we have

$$\text{Prob}[\text{Sample}(\sigma_1) = \pi] = \text{Prob}[\text{Sample}(\sigma_2) = \pi] = \frac{1}{4} \text{ and } \text{Prob}[\text{Sample}(\pi) = \pi] = \frac{1}{2}$$

. Summing up we get $\Sigma_{\delta \in B_n(e,2t)} \text{Prob}[\text{Sample}(\delta) = \pi] = 1$ as desired.

Now suppose $\pi \in C_\tau \setminus C$. Let $\phi_1(\pi) = \psi$. There are two possibilities, first if $\phi_1(\pi) \in C$ then clearly $\phi_1'(\psi) = \pi$. The definition of Sample implies that

$$\sum_{\delta \in B_n(e,2t)} \text{Prob}[\text{Sample}(\delta) = \pi] = \text{Prob}[\text{Sample}(\psi) = \pi] + \text{Prob}[\text{Sample}(\pi) = \pi].$$

So it follows that $\sum_{\delta \in B_n(e,2t)} \text{Prob}[\text{Sample}(\delta) = \pi] = \frac{1}{4} + \frac{3}{4} = 1$. In the second case when $\phi_1(\pi) = \psi \notin C$, we have $\sum_{\delta \in B_n(e,2t)} \text{Prob}[\text{Sample}(\delta) = \pi] = \frac{1}{2} + \frac{1}{2} = 1$. The case when $\pi \in C_{\tau^{-1}} \setminus C$ is similar. This proves the lemma. ∎

It follows from the definition of Sample that replacing τ_i by Sample(τ_i) does not affect the distribution of ψ_i in the Step 2 of the Algorithm 1. In fact, Sample(τ_i) and τ_i are identically distributed by Lemma 3.9. In the Step 1 of the Algorithm 1 we pick each τ_i uniformly at random from $B_n(e, 2t)$. Now, in our analysis we replace this by Sample(τ_i). The crucial point of the argument is that it suffices to replace τ_i by Sample(τ_i) after the Step 2 in the Algorithm 1, as the τ_i is only used to define ψ_i and it will not affect the distribution of ψ_i if we replace τ_i by Sample(τ_i). Note that τ_i is used during sieving

step in the while loop only if i lies in the sieved set S. The remaining τ_i's are replaced by Sample(τ_i) in the Step 5. Clearly, this modification does not change the probability of computing a shortest permutation as the distributions in the two cases are the same. Note that Sample(τ_i) is introduced for analysis. We cannot implement the procedure Sample efficiently as we do not know τ.

In the Step 1 of the Algorithm we pick each τ_i uniformly at random from $B_n(e, 2t)$. The initial set is $\{\tau_i \mid i \in [N]\}$. The while loop iterates for at most $2 \log n$ steps and in each step we remove a set S of size at most $2^{c_1 n}$, where c_1 is given by Lemma 3.3. Thus, at the end of the while loop we still have $N - 2 \log n \cdot 2^{c_1 n}$ many τ_i in the remaining set T. Thus, as argued earlier for the purpose of analysis we can replace τ_i by Sample(τ_i) for all $i \in T$ and it still doesn't affect the working of the algorithm.

The triangle inequality implies $B_n(e, t) \subseteq C_\tau$. Thus Vol$(C_\tau) \geq$ Vol$(B_n(e, t)) \geq t^n \cdot e^{-2n}$ by Lemma 3.3. Also, Vol$(B_n(e, 2t)) \leq (5t)^n$. Hence,

$$\frac{\text{Vol}(C_\tau \cup C_{\tau^{-1}})}{\text{Vol}(B_n(e, 2t))} \geq 2^{-c_2 n},$$

for some constant c_2 (which depends on c_1). Thus a random $\pi \in B_n(e, 2t)$ lies in $C_\tau \cup C_{\tau^{-1}}$ with probability at least $2^{-c_2 n}$.

Given a constant $c_3 > 0$, we can choose a suitably large $N = 2^{cn}$ for a constant c so that at least $2^{c_3 n}$ many τ_i for $i \in T$ at the end of the while loop will lie in $C_\tau \cup C_{\tau^{-1}}$. Thus, with probability $1 - 2^{-\Omega(n)}$ we can guarantee that at least $2^{c_3 n}$ many τ_i for $i \in T$ are such that $\tau_i \in C_\tau \cup C_{\tau^{-1}}$ and $(\varphi_i, \tau_i) \in Z$ at the beginning of the Step 5.

Furthermore, at the beginning of the Step 5 each $(\varphi_i, \tau_i) \in Z$ satisfies $\|\tau_i^{-1}\varphi_i\| \leq 8t$ and $\tau_i^{-1}\varphi_i \in G$. Now we argue using the pigeon-hole principle that there is some $\pi \in G$ such that $\pi = \tau_i^{-1}\varphi_i$, $(\varphi_i, \tau_i) \in Z$ for at least 2^n indices $i \in T$.

Claim 6 $|G \cap B_n(e, 8t)| < 2^{c_4 n}$ *for some constant* c_4.

Proof Note that $l_\infty(\pi_1, \pi_2) \geq t$ for distinct $\pi_1, \pi_2 \in G$. Thus, metric balls of radius $t/2$ around each element in $G \cap B_n(e, 8t)$ are all pairwise disjoint. By triangle inequality,

all these $t/2$ radius metric balls are contained in $B_n(e, 8t + t/2)$. Hence by Lemma 3.3,

$$|G \cap B_n(e, 8t)| < Vol(B_n(e, 17t/2))/Vol(B_n(e, t/2)) < 2^{c_4 n}$$

, This proves the claim. ∎

Let $c_3 = c_4 + 1$. Then with probability $1 - 2^{-\Omega(n)}$ we have $\pi \in G$ such that $\pi = \tau_i^{-1}\varphi_i$, $(\varphi_i, \tau_i) \in Z$ for at least $2^{c_3 n}/2^{c_4 n} = 2^n$ indices $i \in T$. Call this set of indices T_0.

Recall that in our analysis we can replace τ_i by Sample(τ_i) for each $i \in T_0$. By the definition of Sample(τ_i),

$$\text{Prob}[\text{Sample}(\tau_i) = \tau_i \; \forall \; i \in T_0] \leq (3/4)^{2^n}.$$

Similarly,

$$\text{Prob}[\text{Sample}(\tau_i) \neq \tau_i \; \forall \; i \in T_0] \leq (1/2)^{2^n}.$$

Hence with probability $1-2^{-\Omega(n)}$ there are indices $i, j \in T_0$ such that $(\varphi_i, \tau_i), (\varphi_j, \tau_j) \in Z$ and Sample$(\tau_i) = \tau_i$ Sample$(\tau_j) \neq \tau_j$. Clearly, Sample$(\tau_j) = \tau_j\tau$ or Sample$(\tau_j) = \tau_j\tau^{-1}$. Without loss of generality, assume Sample$(\tau_j) = \tau_j\tau$. Then, after the Step 5 of the Algorithm 1 with probability $1 - 2^{-\Omega(n)}$ we have, $\varphi_{i,j} = ((\tau_j\tau)^{-1}\varphi_j)(\tau_i^{-1}\varphi_i)^{-1} = \tau^{-1}\pi\pi^{-1} = \tau^{-1}$. In other words, with probability $1 - 2^{-\Omega(n)}$ one of the $2^{O(n)}$ output permutations is a "shortest" permutation in G. We have shown the following theorem.

Theorem 3.10 *Given a permutation group $G \leq S_n$ as input, there is a randomize $2^{O(n)}$ time algorithm which finds a permutation in $G \setminus \{e\}$ with the smallest possible norm with respect to l_∞ metric with probability at least $1 - 2^{-\Omega(n)}$.*

3.3 Weight Problems for Hamming metric

In this section we consider the weight problems for Hamming metric. As seen earlier the design version of the minimum weight problem(MWP) with respect to Hamming metric is NP-hard [CW06]. Given a permutation group $G \leq S_n$, Maximum Weight Problem is to find $\tau \in G$ of largest possible norm. In the same paper it is shown that the decision

version of the Maximum Weight Problem with respect to Hamming metric is also NP-hard. Interestingly search version of Maximum Weight Problem for ℓ_∞ metric can be solved in polynomial time [CW06].

In this section we give $2^{O(n)}$ algorithm to solve both of these weight problems for Hamming metric.

First we give an easy $2^{O(n)}$ time deterministic algorithm to find $\tau \in G \setminus \{e\}$ with the least possible Hamming norm. It turns out we can use a well-known algorithm from permutation groups. Suppose $G \leq S_n$ is given by a generator set. The problem is to find a shortest permutation in G for the Hamming metric. For every $S \subseteq [n]$ consider the point-wise stabilizer subgroup $G_S \leq G$ defined as $G_S = \{g \in G | \, \forall \, i \in S : \; g(i) = i\}$. Using the Schrier-Sims algorithm in polynomial time [Lu93] we can compute a generating set for G_S. Thus, in $2^{O(n)}$ time we can compute G_S for all $S \subseteq [n]$ and find the largest $t < n$ for which there is $S \subseteq [n]$ such that $|S| = t$ and G_S is a nontrivial subgroup. Clearly, any $\tau \neq e \in G_S$ is a shortest permutation with respect to Hamming metric.

Maximum Weight Problem: First we consider a special case of Maximum Weight Problem. Given permutation group $G \leq S_n$ goal is to check whether G has a fixed-point free permutation (i.e. a permutation with Hamming norm n), and if G has a fixed-point free permutation output one. Using Inclusion-Exclusion Principle we give a $2^{O(n)}$ time deterministic algorithm for the search version of the problem.

As before, let G_S be the subgroup of G that point-wise fixes $S \subseteq [n]$. Let $F \subseteq G$ denote the set of fix-point free elements. Clearly, $F \cap G_S = \emptyset$ for each nonempty S. Also, $F \cup \bigcup_{S \neq \emptyset} G_S = G$. In $2^{O(n)}$ time we can compute generating sets for all G_S.

Consider the cosets of $G_{[1]}$ inside G. Clearly if G has a fixed-point free permutation τ, $\tau \notin G_{[1]}$ and τ lies in one of the cosets of $G_{[1]}$. Basic idea of the algorithm is to search a fixed-point free permutation in these coset individually. We know that for a set of permutations in the same coset of $G_{[1]}$, 1 is mapped to a fixed element. This enable us to successively fix images of $1, \ldots, k$ and work with smaller and smaller cosets.

So, inductively assume that we have already computed a coset H_{k-1} of $G_{[k-1]}$ in G,

where for all $\tau \in H_{k-1}$ we have $\tau(i) = \alpha_i$, $\alpha_i \in [n]$, $\alpha_i \neq i$ for $1 \leq i \leq k-1$ and H_{k-1} contains a fix-point free permutation if G does.

We now show how to compute a point $\alpha_k \in [n]$ which will fix the coset H_k of $G_{[k]}$ in $2^{O(n)}$ time such that for all $\tau \in H_k$, $\tau(i) = \alpha_i$, $\alpha_i \neq i$ for $i = 1$ to k and H_k contains a fixed-point free element if G contains a one. It is easy to see that by repeating this successively we can find a fix-point free permutation in G.

First, from the orbit of k under action of G we pick a candidate point α_k distinct from $\alpha_1, \cdots, \alpha_{k-1}$ and k. Let $H_k = \{\tau \in G | \tau(i) = \alpha_i, 1 \leq i \leq k\}$.

Let $A_i = H_k \cap G_{\{i\}}$ for $i = k+1$ to n. It is clear that H_k contains a fixed point free permutation iff $A_{k+1} \cup A_{k+2} \cup ... \cup A_n \subset H_k$ iff $|A_{k+1} \cup A_k + 2 \cup ... \cup A_n| < |H_k|$. So if we can compute $|A_{k+1} \cup A_{k+2} \cup ... \cup A_n|$ we can simply compare with $|H_k|$ and if it is strictly less than $|H_k|$ we know that there is a fixed-point free permutation in H_k. So we have found α_k such that for all $\tau \in H_k$, $\tau(i) = \alpha_i$, $\alpha_i \neq i$ for $i = 1$ to k and H_k contains a fix-point free element if G contains a one. If $|H_k| = |A_{k+1} \cup \ldots \cup A_n|$, then there is no fixed point free permutation in H_k for the current choice of α_k, so we pick another candidate value for α_k in the orbit of k and proceed similarly.

To summarize, the question boils down to computing cardinality of $A_{k+1} \cup \ldots \cup A_n$, for which we are going to use inclusion-exclusion principle. By IEP we know that,

$$|A_{k+1} \cup \ldots \cup A_n| = \sum_{S \subseteq \{k+1,\ldots,n\}} (-1)^{|S|+1} |\cap_{j \in S} A_j|.$$

To compute right hand side of the above equation we need to know $|\cap_{j \in S} A_j|$ for all subsets $S \subseteq \{k+1, \ldots, n\}$. From the definition of A_i's it is clear that $|\cap_{j \in S} A_j| = H_k \cap G_S$ for any set $S \subseteq \{k+1, \ldots, n\}$. We can compute a generating set for G_S in polynomial time for any $S \subseteq \{k+1, \ldots, n\}$ using the Schrier-Sims algorithm [Lu93]. Furthermore, the coset intersection problem $H_k \cap G_S$ can also be solved in $2^{O(n)}$ time using results of Babai and Luks [BL83, Lu93]. Thus, in time $2^{O(n)}$ we can compute $|\bigcap_{i \in S} A_i|$ for all subsets $S \subseteq \{k+1, \ldots, n\}$. In $2^{O(n)}$ further steps, by using the Inclusion-Exclusion formula, we can compute $|A_{k+1} \cup A_{k+2} \cup \ldots \cup A_n|$.

This gives $2^{O(n)}$ time algorithm to find a fixpoint free permutation.

The algorithm for Maximum Weight Problem is similar only with some minor changes to the algorithm for computing fixed-point free permutation. We briefly describe it below. Let $G \leq S_n$ is a given group we want to compute $\tau \in G$ with maximum possible Hamming norm. Consider pointwise stabilizer groups G_S for all $S \subseteq [n]$. For each S we compute a fixed-point free permutation in G_S (if one exists) and output a largest Hamming norm permutation among these. Correctness of the algorithm is almost immediate. We summarize results in this section in the following theorem.

Theorem 3.11 *Given a permutation group $G \leq S_n$ by a generating set, in $2^{O(n)}$ time we can find $\tau \in G \setminus \{e\}$ with the smallest possible norm and $\psi \in G$ with the largest possible norm with respect to Hamming metric.*

3.4 MWP is reducible to SDP for solvable permutation groups

For integer lattices, SVP (shortest vector problem) is polynomial-time reducible to CVP (closest vector problem) [GMSS99]. A similar result for linear codes is also proved there. We show an analogous result for *solvable* permutation groups. In fact we give a polynomial-time Turing reduction from MWP to SDP, which works for the gap version of the problem for *any* right invariant metric d. We do not know if this reduction can be extended to non-solvable permutation groups. Finally we make an observation about the hardness of approximation of SDP and MWP.

Let $G \leq S_n$ be input instance for MWP. The idea is to make different queries of the form (H, τ) to SDP, for suitable subgroups $H \leq G$ and $\tau \notin H$.

Let d be a right-invariant metric on S_n. We want to find a shortest permutation $\tau \in G$ w.r.t. metric d. It is well-known in algorithmic permutation group theory (e.g. see [Lu93]) that for solvable permutation groups $G \leq S_n$ we can compute in deterministic polynomial time a composition series $G = G_k \rhd G_{k-1} \rhd \ldots \rhd G_1 \rhd G_0 = \{e\}$, $k \leq n$. In other words, G_{i-1} is a normal subgroup of G_i for each i. Furthermore, since G is solvable, each quotient group G_i/G_{i-1} has prime order, say p_i (where the p_i's need not be distinct). Notice that for any $\tau_i \in G_i \setminus G_{i-1}$, the coset $G_{i-1}\tau_i$ generates the cyclic

quotient group G_i/G_{i-1}. It is easily seen that these elements τ_i form a generating set for G with the following standard property.

Proposition 3.12 *For each $i, 1 \leq i \leq k$, every $\tau \in G_i \setminus G_{i-1}$ can be uniquely expressed as $\tau = \tau_1^{\alpha_1}\tau_2^{\alpha_2}\ldots\tau_i^{\alpha_i}, 0 \leq \alpha_j < p_j, 1 \leq j \leq i$ and $\alpha_i \neq 0$.*

Proof We prove the claim by induction on i. For $i = 1$, order of $G_1/\{e\}$ is a prime p_1, hence G_1 is cyclic group of order p_1. Therefore every $\tau \in G_1 \setminus \{e\}$ can be uniquely expressed as $\tau_1^{\alpha_1}, 0 < \alpha_1 < p_1$. Assume the claim is true for all $t \leq i-1$. We know that $\tau_i \in G_i \setminus G_{i-1}$ and order of G_i/G_{i-1} is prime p_i. Therefore every right cosets of G_{i-1} in G_i can be uniquely written as $G_{i-1}\tau_i^j$, $0 \leq j < p_j$. Hence $G_i = G_{i-1} \cup G_{i-1}\tau_i \cup \ldots \cup G_{i-1}\tau_i^{p_i-1}$ where $\cup$ denotes disjoint union. Now using induction hypothesis the claim follows. ∎

Theorem 3.13 *For any right invariant metric d on S_n, and for solvable groups,* GapMWP$_\gamma$ *is polynomial time Turing reducible to* GapSDP$_\gamma$.

Proof Let (G, m) be an input instance of GapMWP$_\gamma$. We compute $\tau_1, \ldots \tau_k$ for the group G as described above. Then we query the oracle of GapSDP$_\gamma$ for instances $(G_{i-1}, \tau_i^{-r}, m)$, for $1 \leq i \leq k, 1 \leq r < p_i$. The reduction outputs "YES" if at least one of the queries answers "YES" otherwise it outputs "NO". Clearly, the reduction makes at most $O(n^2)$ oracle queries and runs in polynomial time. We prove its correctness.

Suppose (G, m) is a "YES" instance of GapMWP$_\gamma$. We show that at least one of the queries $(G_{i-1}, \tau_i^{-r}, m), 1 \leq i \leq k, 1 \leq r < p_i$ will return "YES". Let $\tau \in G = G_k$ such that $\|\tau\| \leq m$. Let i be the smallest such that $\tau \notin G_{i-1}$, $\tau \in G_i$. From Proposition 3.12 it follows that τ can be uniquely expressed as $\prod_{j=1}^{i} \tau_j^{\alpha_j}$, where $0 \leq \alpha_j < p_j, 1 \leq j \leq i$ and $\alpha_i \neq 0$. As $\prod_{j=1}^{i-1} \tau_j^{\alpha_j} \in G_{i-1}$, we have $d(\tau_i^{-\alpha_i}, G_{i-1}) \leq d(\tau_i^{-\alpha_i}, \prod_{j=1}^{i-1} \tau_j^{\alpha_j})$. The right invariance of d implies $d(\tau_i^{-\alpha_i}, G_{i-1}) \leq d(e, \prod_{j=1}^{i} \tau_j^{\alpha_j}) = \|\tau\| \leq m$. Hence $(G_{i-1}, \tau_i^{-\alpha_i}, m)$ is a "YES" instance of GapSDP$_\gamma$.

Now suppose $(G_{i-1}, \tau_i^{-r}, m), 1 \leq i \leq k$,$1 \leq r \leq p_i - 1$ is not a "NO" instance of GapSDP$_\gamma$. Then there is some $\tau \in G_{i-1}$ such that $d(\tau, \tau_i^{-r}) \leq \gamma m$, i.e. $\|\tau\tau_i^r\| \leq \gamma m$. As $\tau_i \in G_i \setminus G_{i-1}$, $\tau_i^t \notin G_{i-1}$ for $1 \leq t \leq p_i - 1$. Thus $\tau_i^r \notin G_{i-1}$ implying $\tau\tau_i^r \neq e$. Hence (G, m) is not a "NO" instance of GapMWP$_\gamma$. This completes the proof. ∎

Cameron et al [BCW06, CW06] have shown that SDP and MWP are NP-hard for several permutation metrics. It follows from [ABSS97] that SDP for linear codes is NP-hard to approximate within a factor of $(\log n)^c$, where n is the block length of the input code and c is an arbitrary constant. Furthermore, Dumer et al [DMS99] have shown that constant-factor approximation is NP-hard for MWP restricted to binary linear codes. Given a binary linear code C of block length n, we can easily construct an abelian 2-group $G \leq S_{2n}$ isomorphic to C. We associate vector $(a_1, \ldots, a_n) \in C$ to a permutation $\tau \in S_{2n}$ such that in product of cycle notation $\tau = (1,2)^{a_1}(3,4)^{a_2}...(2n-1,2n)^{a_n}$, where $(i, i+1)$ denotes a 2-cycle. It is clear that C is isomorphic to G and the Hamming weight of any permutation $\tau \in G$ is two times as that of the Hamming norm of the associated vector $(a_1, \ldots, a_n) \in C$. An easy consequence of this construction and known hardness results for binary linear codes directly yields the following hardness results for GapSDP$_\gamma$ and GapMWP$_\gamma$ for Hamming metric. We can also give a similar construction which works for Cayley metric. For the details of the construction in the case of Cayley metric we refer to [CW06].

Theorem 3.14 *For Hamming and Cayley metric* GapSDP$_\gamma$ *is NP-hard for* γ *equal to* $O((\log n)^c)$ *and* GapMWP$_\gamma$ *is NP-hard under randomized reduction for any constant* γ.

3.5 Limits of hardness

Since GapSDP$_\gamma$ is NP-hard for $\gamma \leq (\log n)^c$, a natural question is to explore its complexity for larger gaps. For the GapCVP problem on lattices, Goldreich and Goldwasser [GG00] have shown a constant round IP protocol for $O(\sqrt{n/\log n})$ gap in the case of l_2 norm. Consequently, for this gap GapCVP is not NP-hard unless polynomial hierarchy collapses. We adapt similar ideas to the permutation group setting. For the Hamming and Cayley metric we give a constant round IP protocol for the complement problem of GapSDP$_\gamma$ for $\gamma \geq n/\log n$, such that the protocol rejects "YES" instances of GapSDP$_\gamma$ with probability at least $n^{-\log n}$, and always accepts the "NO" instances. Note that there is no specific reason for choosing Hamming or Cayley metrics. Actually the protocol is fairly generic, it needs certain volume bounds on metric balls, right invariance of the metric and uniform sampling procedure from metric balls. So it might work for other

metrics as-well, we have chosen these metrics only as a representative.

For designing the IP protocols we require uniform random sampling procedures from metric balls for the Hamming and Cayley metrics.

We first consider the Cayley metric. Recall that the Cayley distance between τ and e is the least number of transpositions required to take τ to e. Let k be the number of cycles in τ. Each transposition multiplied to τ increases or decreases the number of cycles by 1. Since τ is transformed to e with the fewest transpositions if we always multiply by a transposition that increments the number of cycles, we have $d(e, \tau) = n - k$. Thus, a Cayley metric ball of radius r contains $\tau \in S_n$ such that τ has at least $n - r$ cycles. The number $c(n, k)$ of permutation in S_n with exactly k cycles is a Stirling number of the first kind and it satisfies the recurrence relation $c(n, k) = (n-1)c(n-1, k)+c(n-1, k-1)$. We can compute $c(m, l)$, $0 \leq m \leq n, 0 \leq l \leq k$ using the recurrence for $c(n, k)$.

Proposition 3.15 *Let $S \subseteq S_n$ be the set of permutations with k cycles. Let $N = |S| = c(n, k)$. Then there exists a polynomial (in n) time computable bijective function $f_{n,k} : [N] \mapsto S$.*

Proof If $n = k = 1$, clearly such function exists, $f_{1,1}(1)$ is simply defined as identity element of S_1. We use induction on $n+k$. Assume that such functions exist for $n+k \leq t$. Now consider n, k such that $n+k = t+1$. We define the function $f_{n,k}(i)$, for $1 \leq i \leq N$:

1. If $i > (n-1)c(n-1, k)$, let $\pi = f_{n-1,k-1}(i - (n-1)c(n-1, k))$ and τ be obtained by appending a 1-cycle (n) to π. Define $f_{n,k}(i) = \tau$.
2. If $i \leq (n-1)c(n-1, k)$ then find j such that $(j-1)c(n-1, k) < i \leq jc(n-1, k)$. Let $\pi = f_{n-1,k}(i - (j-1)c(n-1, k))$, write π as product of disjoint cycles. Let $\tau \in S_n$ be obtained by inserting n in the j^{th} position of the cyclic decomposition of π. Define $f_{n,k}(i) = \tau$.

Clearly, $f_{n,k}$ is polynomial time computable. We show $f_{n,k}$ is bijective by induction. Suppose $f_{n-1,k-1}$ and $f_{n-1,k}$ are bijective. Each $\tau \in S_n$ with k cycles can be uniquely obtained either by inserting element n in cyclic decomposition of a $\pi \in S_{n-1}$ with k

cycles (which can be done in $n-1$ ways) or by attaching a 1-cycle with element n to some $\pi \in S_{n-1}$ with $k-1$ cycles. It follows that $f_{n,k}$ is bijective. ∎

To uniformly sample $\tau \in S_n$ with k cycles, we pick $m \in \{1, 2, \ldots, c(n,k)\}$ uniformly at random and let $\tau = f_{n,k}(m)$.

Lemma 3.16 *There is a randomized procedure which runs in time $poly(n)$ and samples from $B_n(e, r, d)$ uniformly, where d denotes Cayley metric.*

Now consider the Hamming Metric. The Hamming ball of radius r contains all $\tau \in S_n$ such that $\tau(i) \neq i$ for at most r many points i. Hence, $\text{Vol}(B_n(e, r, d)) = \Sigma_{i=0}^{r} \binom{n}{i} D_i$, where D_i denotes the number of derangements on i points. We can easily enumerate all i-element subsets of $[n]$. The number D_i of derangements on i points satisfies the recurrence $D_i = (i-1)(D_{i-1} + D_{i-2})$. With similar ideas as used for sampling for Cayley metric balls we can do uniform random sampling from Hamming metric balls in polynomial time.

Lemma 3.17 *For $r > 0$, there exists a randomized procedure which runs in time $poly(n)$ and samples uniformly at random from the Hamming balls of radius r around e ($B_n(e, r, d)$).*

We now describe the simple 2-round IP protocol for the Hamming metric. Let (G, τ, r) be input instance of GapSDP$_\gamma$ for $\gamma \geq n/\log n$, and d is the Hamming metric.

1. **Verifier:** picks $\sigma \in \{0,1\}, \psi \in G, \beta \in B_n(e, \gamma r/2, d)$ uniformly at random. The verifier sends to the prover the permutation $\pi = \beta\psi$ if $\sigma = 0$, and $\pi = \beta\tau\psi$ if $\sigma = 1$.
2. **Prover:** The prover sends $b = 0$ if $d(\pi, G) < d(\pi, \tau G)$ and $b = 1$ otherwise.
3. **Verifier:** Accepts iff $b = \sigma$.

For the protocol we need polynomial time random sampling from a permutation group which is quite standard [Lu93]. We also need uniform sampling from Hamming metric

balls which is given by Lemma 3.17. To prove correctness of the protocol, we need following Proposition.

Proposition 3.18 *If $d(\tau, G) > \gamma r$ then for all $\psi_1, \psi_2 \in G$,*

$$B_n(\psi_1, \gamma r/2, d) \cap B_n(\tau\psi_2, \gamma r/2, d) = \emptyset$$

Proof Suppose $\pi \in B_n(\psi_1, \gamma r/2, d) \cap B_n(\tau\psi_2, \gamma r/2, d)$. We have $d(\psi_1, \pi) \leq \gamma r/2$ and $d(\pi, \tau\psi_2) \leq \gamma r/2$. By triangle inequality, $d(\psi_1, \tau\psi_2) \leq \gamma r$. This implies $d(\psi_1\psi_2^{-1}, \tau) \leq \gamma r$. But $d(\psi_1\psi_2^{-1}, \tau) \geq d(\tau, G) > \gamma r$, a contradiction. This proves the proposition. ∎

Lemma 3.19 *The verifier always accepts if (G, τ, r) is "NO" instance of* GapSDP$_\gamma$ *. Furthermore, the verifier rejects with probability at least $n^{-\log n}$ if (G, τ, r) is a "YES" instance of* GapSDP$_\gamma$ *.*

Proof Let (G, τ, r) is "NO" instance of GapSDP$_\gamma$. That means, $d(\tau, G) > \gamma r$. Suppose $\sigma = 0$. Then $d(\pi, G) \leq d(\beta\psi, \psi) = d(\beta, e) \leq \gamma r/2$. Hence $\pi \in B_n(\psi, \gamma r/2, d)$, $\psi \in G$. It follows from Proposition 3.18 that, for all $\psi_2 \in G$ we have $\pi \notin B_n(\tau\psi_2, \gamma r/2, d)$. That implies $d(\pi, \tau G) > \gamma r/2 \geq d(\pi, G)$, implying the prover responses with 0. For $\sigma = 1$ the proof is similar.

The above argument uses only right invariance of the metric d and it works for any gap γ.

To prove the soundness of the protocol, for any (G, τ, r) that is a "YES" instance of GapSDP$_\gamma$ and any prover, we will show that the verifier accepts with probability at most $1 - n^{-\log n}$. Notice that we may assume $r \leq \log n$. For, if $r > \log n$ then $\gamma r > n$ and the verifier can always reject such an instance. Thus, $d(\tau, G) \leq r \leq \log n$. Hence, there is a $\rho \in G$ such that $d(\tau, \rho) \leq r$.

Let D_0 and D_1 denote the distributions of the verifier's first round message for $\sigma = 0$ and $\sigma = 1$ respectively. To prove soundness, it suffices to show that the statistical difference between D_0 and D_1 is bounded by $1 - n^{-\log n}$. In D_0, the permutation sent by verifier is uniformly distributed in a ball of radius $\gamma r/2$ around ψ for $\psi \in_R G$. In D_1, the permutation is uniformly distributed in a ball of radius $\gamma r/2$ around $\tau\psi$ for $\psi \in_R G$.

For any $\psi \in G$, let $\phi = \rho\psi$ and consider balls of radius $\gamma r/2$ around ϕ and $\tau\psi$. To prove the soundness it is sufficient to argue $\text{Vol}(B_n(\phi, R, d) \cap B_n(\tau\psi, R, d)) \geq n^{-\log n} Vol(B_n(e, R, d))$, where the radius $R = \gamma r/2$. Clearly $R \leq n/2$. We have $\phi = \rho\psi \in G$ and $d(\phi, \tau\psi) = d(\rho\psi, \tau\psi) = d(\rho, \tau) \leq r$. By triangle inequality, we have $B_n(\phi, R - r, d) \subseteq B_n(\phi, R, d) \cap B_n(\tau\psi, R, d)$. By right invariance of Hamming metric, it suffices to prove $\text{Vol}(B_n(\phi, R - r, d)) = \text{Vol}(B_n(e, R - r, d)) \geq n^{-\log n}\text{Vol}(B_n(e, R, d))$.

Claim 7 *Vol*$(B_n(e, R - r, d)) \geq n^{-\log n}$*Vol*$(B_n(e, R, d))$*, where* d *is Hamming metric and* $r \leq \log n$*,* $R \leq n/2$*.*

To see the claim notice that we have

$$\text{Vol}(B_n(e, R - r, d)) = \Sigma_{i=0}^{i=R-r} \binom{n}{i} D_i \approx \Sigma_{i=0}^{i=R-r} \frac{n!}{(n-i)!i!} \frac{i!}{e} > \frac{n!}{e(n-R+r)!}.$$

Similarly,

$$\text{Vol}(B_n(e, R, d)) = \Sigma_{i=0}^{i=R} \binom{n}{i} D_i \approx \Sigma_{i=0}^{i=R} \frac{n!}{(n-i)!i!} \frac{i!}{e} < (R+1) \frac{n!}{e(n-R)!}.$$

Combining we get

$$\text{Vol}(B_n(e, R - r, d)) \geq \frac{R+1}{(n-R+r)^r} \text{Vol}(B_n(e, R, d)) \geq \frac{\text{Vol}(B_n(e, R, d))}{n^{O(\log n)}}.$$

∎

This shows the correctness of the protocol for Hamming metric. For the Cayley metric too a similar IP protocol can be designed. As an immediate consequence we have the following.

Corollary 3.20 *For the Hamming and Cayley metrics,* GapSDP$_\gamma$ *for* $\gamma \geq n/\log n$ *is not NP-hard unless coNP has constant round interactive protocols with constant error probability with the verifier allowed* $n^{O(\log n)}$ *running time.*

Recall that GapMWP$_\gamma$ is Turing reducible to GapSDP$_\gamma$ for solvable groups by Theorem 3.13 and the Turing reduction makes queries with the same gap. Hence, by the above corollary it follows that GapMWP$_\gamma$ w.r.t. solvable groups for $\gamma > n/\log n$ is also unlikely to be NP-hard for Hamming and Cayley metrics.

3.6 Overview

Our goal in this chapter was to study the Minimum Weight Problem and the Subgroup Distance Problem for permutation groups. Interestingly we can adapt upper and lower bound results from the analogous problems in case of integer lattices.

We studied the algorithmic complexity of MWP with respect to Hamming and ℓ_∞ metrics. It is known that MWP is NP-complete for several natural permutation metrics including Hamming and ℓ_∞ metric, even if the concerned permutation group is abelian. If the given group is an abelian permutation group then its size is bounded by $2^{O(n)}$. So both the problems MWP and SDP can be solved in $2^{O(n)}$ time for abelian permutation groups by enumerating the elements of given group. More non-trivial case is the case of non-abelian permutation groups.

We gave a $2^{O(n)}$ time algorithm for MWP in case of Hamming metric. Our algorithm is group theoretic and is based on the classical Schrier-Sims algorithm. MWP with respect to ℓ_∞ metric does not appear amenable to a permutation group-theoretic approach. We gave a randomized $2^{O(n)}$ time algorithm for MWP with respect to ℓ_∞ metric. Our algorithm adapts ideas from Ajtai-Kumar-Sivakumar sieving algorithm for shortest vector problem for integer lattices. We need to modify the sampling and the perturbation process in the AKS, since the nice geometrical structure is missing in case of permutation groups.

It is known that SDP is NP-hard([BCW06]) and it easily follows that SDP is hard to approximate within a factor of $\log^{O(1)} n$ unless P=NP. In contrast, we showed that SDP for approximation factor more than $n/\log n$ is not NP-hard unless there is an unlikely containment of complexity classes. For several permutation metrics, we showed that the minimum weight problem is polynomial-time reducible to the subgroup distance problem for *solvable* permutation groups. These results adapts ideas from the analogous

results in the case of integer lattices.

4

Arithmetic Circuits, Branching Programs and Monomial Algebras

In this chapter we study arithmetic circuit and algebraic branching program size lower bound questions as well as polynomial identity testing problem over *monomial algebras* both in the noncommutative and the commutative setting.

We also study the Monomial Search Problem, which is a natural search version of the Polynomial Identity Testing Problem. We mainly explore the randomized parallel complexity of the Monomial Search Problem.

Proving superpolynomial size lower bounds for the commutative arithmetic circuits and algebraic branching programs computing explicit polynomials is a challenging problem in complexity theory. Such explicit size lower bounds are known only for some special commutative arithmetic circuit classes, like depth 3 circuits and some restricted classes of depth 4 circuits.

In the noncommutative setting the question is better understood. Nisan in his pioneering paper [N91] studied the lower bounds for noncommutative computation. Using a rank argument Nisan showed that the Permanent and the Determinant polynomials in the *free* noncommutative ring $\mathbb{F}\{x_{11}, \cdots, x_{nn}\}$ require exponential size noncommutative formulas (and noncommutative algebraic branching programs). Chien and Sinclair [CS04] explored the same question over other noncommutative algebras. They refined Nisan's rank argument to show exponential size lower bounds for formulas computing

the Permanent or the Determinant over the algebra of 2×2 matrices over $\mathbb{F}$, the quaternion algebra, and several other interesting examples. In similar spirit as [CS04], we explore the lower bound question over other noncommutative algebras.

Recall that an *ideal* I of the noncommutative polynomial ring $\mathbb{F}\{X\}$ is a subring that is closed under both left and right multiplication by ring elements. The circuit complexity of the polynomial f in the the quotient algebra $\mathbb{F}\{X\}/I$ is $C_I(f) = \min_{g \in I} C(f+g)$ where for $h \in \mathbb{F}\{X\}$, $C(h)$ is the circuit complexity of h over free noncommutative algebra $\mathbb{F}\{X\}$. We can define the algebraic branching program complexity of a polynomial over $\mathbb{F}\{X\}/I$ analogously. Broadly speaking, our goal is to study the question of proving lower bound on explicit polynomials over quotient algebra $\mathbb{F}\{X\}/I$ where the ideal I is given by a generating set of polynomials. If the ideal I is generated by monomials in $\mathbb{F}\{X\}$ the algebra $\mathbb{F}\{X\}/I$ is called as monomial algebra. We will focus on monomial algebras in this chapter.

It turns out that the structure of monomial algebras is intimately connected with automata theory. Suppose $X = \{x_1, x_2, \cdots, x_n\}$ is a set of n noncommuting variables. Let $\mathbb{F}\{x_1, x_2, \cdots, x_n\}$ denote the free noncommutative polynomial ring generated by the variables in X over a field $\mathbb{F}$. The polynomials in this algebra are $\mathbb{F}$-linear combinations of words over X. Given an arithmetic circuit or an algebraic branching program computing polynomial $f \in \mathbb{F}\{X\}$ and an automaton $\mathcal{A}$ accepting some language $L(\mathcal{A})$ over the alphabet X, we will define the notions of *intersecting* and *quotienting* circuit or an ABP by an automaton.

Informally, these are filtering operations that allow us to define new polynomials from a given polynomial. Thus, given a polynomial f, we can efficiently construct a circuit(or ABP) computing polynomial g, such that $\text{mon}(g) = \text{mon}(f) \cap L(\mathcal{A})$ or $\text{mon}(g) = \text{mon}(f) \backslash L(\mathcal{A})$ where $\text{mon}(f)$, $\text{mon}(g)$ denotes the set of monomials of f, g respectively.

If I is a *finitely generated monomial ideal* of $\mathbb{F}\{X\}$, we can design a polynomial-size "pattern matching" DFA $\mathcal{A}$ that accepts precisely the monomials in I. Using the notion of quotient by an automata we can reduce the problem of proving lower bounds (and polynomial identity testing) for the monomial algebra $\mathbb{F}\{X\}/I$ to the free noncommutative ring $\mathbb{F}\{X\}$. Applying this idea, we show that the $n \times n$ Permanent (and Determinant) in the quotient algebra $\mathbb{F}\{X\}/I$ still requires $2^{\Omega(n)}$ size ABPs if the ideal

I is generated by $2^{o(n)}$ many monomials. Hence, we can extend Nisan's lower bound argument to noncommutative monomial algebras. Furthermore, the Raz-Shpilka deterministic identity test for noncommutative ABPs [RS05] also carry over to $\mathbb{F}\{X\}/I$.

In the commutative setting, Jerrum and Snir [JS82] have shown a $2^{\Omega(n)}$ size lower bound for monotone arithmetic circuits computing the $n \times n$ Permanent. We examine the size of monotone arithmetic circuits for any commutative monomial algebra $\mathbb{Q}[x_{11}, x_{12}, \cdots, x_{nn}]/I$ where I is a monomial ideal. Here as well our tools are automata theoretic. We define notion of *commutative* automata and consider intersection and quotients of circuits by commutative automatons. Our main result here is a $2^{\Omega(n)}$ lower bound for the $n \times n$ Permanent over $\mathbb{Q}[x_{11}, x_{12}, \cdots, x_{nn}]/I$, where the monomial ideal I is generated by $o(n/\log n)$ monomials.

Next, we study the *Monomial Search Problem*. This is a natural search version of polynomial identity testing: Given a polynomial $f \in \mathbb{F}\{X\}$ (or, in the commutative case $f \in \mathbb{F}[X]$) of total degree d by an arithmetic circuit or an ABP, the problem is to *find* a nonzero monomial of the polynomial f. Applying our results on intersection of noncommutative ABPs over $\mathbb{F}$ with a DFA, we give a randomized NC^2 algorithm for finding a nonzero monomial and its coefficient.

We also obtain randomized NC^2 Monomial search algorithm for *commutative* ABPs. For general arithmetic circuits we obtain a randomized NC reduction from the monomial search problem to the identity testing problem.

4.1 Preliminaries

We start with some basic definitions. In this chapter we deal with both commutative and the noncommutative polynomial rings. We denote the commutative polynomial ring over field $\mathbb{F}$ in indeterminates $x_1, \ldots, x_n$ by $\mathbb{F}[x_1, \ldots, x_n]$. Similarly we denote polynomial ring over field $\mathbb{F}$ in free noncommuting variable $x_1, \ldots, x_n$ by $\mathbb{F}\{x_1, \ldots, x_n\}$. Here, by noncommuting variables we mean $x_i x_j - x_j x_i \neq 0$ if $i \neq j$. Sometime we use the notation $\mathbb{F}[X]$ and $\mathbb{F}\{X\}$ to denote these rings if the set of variables X is clear from the context. For polynomial $f \in \mathbb{F}[X]$ (or $f \in \mathbb{F}\{X\}$) mon(f) denotes the set of nonzero monomials of f in the respective ring.

Next we recall some basic definitions from complexity theory.

Definition 4.1 *A* commutative arithmetic circuit *computing a polynomial in the ring* $\mathbb{F}[x_1, \cdots, x_n]$ *is a directed acyclic graph. Each node of in-degree zero is labelled by a variable* x_i *or a field element. Each internal node of the circuit has in-degree* 2 *and is either a* $+$ *(addition gate) or a* $*$ *(multiplication gate). The circuit has one special node designated the output gate which computes a polynomial in* $\mathbb{F}[x_1, \cdots, x_n]$.

A *noncommutative* arithmetic circuit is defined as above except that the inputs to each multiplication gate of the circuit are ordered as left and right (to capture the fact that $*$ is noncommutative). Clearly, such an arithmetic circuit computes a polynomial in the noncommutative ring $\mathbb{F}\{x_1, \ldots x_n\}$.

An arithmetic circuit over field of rationals is called monotone if all the rational numbers labeling leaf nodes of circuit are positive.

Definition 4.2 [N91, RS05] *An* algebraic branching program *(ABP) is a layered directed acyclic graph with one* source *vertex of in-degree zero and one* sink *vertex of out-degree zero. The vertices of the graph are partitioned into layers numbered* $0, 1, \cdots, d$. *Edges may only go from layer* i *to* $i+1$ *for* $i \in \{0, \cdots, d-1\}$. *The source is the only vertex at layer* 0 *and the sink is the only vertex at layer* d. *Each edge is labeled with a homogeneous linear form in the input variables. The size of the ABP is the number of vertices.*

An ABP computes a polynomial in the obvious way: the sum over all paths from the source to the sink, of the product of the linear forms by which the edges of paths are labeled. It is clear that ABP computes degree d homogeneous polynomial.

Next we recall the complexity measures for a polynomial from Nisan [N91]. An arithmetic circuit complexity of a polynomial $f \in \mathbb{F}[X]$ (or $f \in \mathbb{F}\{X\}$) is a size of a smallest arithmetic circuit over $\mathbb{F}[X]$ (or $\mathbb{F}\{X\}$) computing f and it is denoted by $C(f)$. Analogously the algebraic branching program complexity of a polynomial f is defined and it is denoted by $B(f)$. For a polynomial f over rationals with positive coefficients, its

monotone circuit complexity is a size of a smallest monotone circuit computing f and it is denoted by $C^+(f)$.

We now recall definitions of some complexity classes needed for this chapter. Fix a finite input alphabet Σ. A language $L \subseteq \Sigma^*$ is in the class *logspace* (denoted L) if there is a deterministic Turing machine with a read-only input tape and an $O(\log n)$ space-bounded work tape that accepts the language L.

Definition 4.3 *The complexity class* GapL *is the class of functions $f : \Sigma^* \to \mathbb{Z}$, for which there is a logspace bounded nondeterministic Turing machine M such that on any input $x \in \Sigma^*$, we have $f(x) = acc_M(x) - rej_M(x)$, where $acc_M(x)$ and $rej_M(x)$ denote the number of accepting and rejecting computation paths of M on input x respectively.*

A language L is in randomized NC^2 if there is a logspace uniform boolean circuit family $\{C_n\}_{n\geq 1}$ of polynomial size and $\log^2 n$ depth with gates of constant-fanin such that for $x \in \Sigma^n$ we have $\Pr_w[C_n(x,w) = \chi_L(x)] \geq 2/3$, where χ_L is the characteristic function for L.

4.2 Intersecting and Quotienting by Automata

We now define the notions of intersection and quotient by finite automata, and make some observations.

Definition 4.4 *Let $f \in \mathbb{F}\{X\}$ be a polynomial and $\mathcal{A}$ be a finite automaton (deterministic or nondeterministic) accepting a subset of X^*. The* intersection *of $f = \sum c_m m$ by $\mathcal{A}$ is the polynomial $f_{\mathcal{A}} = \sum_{m \in mon(f) \cap L(\mathcal{A})} c_m m$, sometimes we also denote the polynomial $f_{\mathcal{A}}$ by $f(\mathrm{div}\mathcal{A})$.*

Let $f(mod\ \mathcal{A})$ denote the polynomial $f - f_{\mathcal{A}}$. We refer to $f(mod\ \mathcal{A})$ as the quotient *of f by $\mathcal{A}$. Thus, the automaton $\mathcal{A}$ splits the polynomial f into two parts as $f = f(mod\ \mathcal{A}) + f_{\mathcal{A}}$.*

Given an arithmetic circuit C (or an ABP P) computing a polynomial in $\mathbb{F}\{X\}$ and a deterministic finite automaton $\mathcal{A}$ (a DFA or an NFA) we can talk of the polynomials $C_{\mathcal{A}}$,

$C(\mod \mathcal{A})$, $P_{\mathcal{A}}$ and $P(\mod \mathcal{A})$. We focus on the circuit and ABP size complexities of $f(\text{mod } \mathcal{A})$ and $f_{\mathcal{A}}$, in terms of the circuit (resp. ABP) complexity of f and the size of the automaton $\mathcal{A}$, in the case when $\mathcal{A}$ is a deterministic finite automaton. The bounds we obtain are *constructive*: we will efficiently compute a circuit (resp. ABP) for $f(\text{mod } \mathcal{A})$ and $f_{\mathcal{A}}$ from the given circuit (resp. ABP) for f and $\mathcal{A}$. We have the following theorem relating the complexity of $f(\text{mod } \mathcal{A})$ and $f_{\mathcal{A}}$ to the complexity of f.

Theorem 4.5 *Let $f \in \mathbb{F}\{X\}$ and $\mathcal{A}$ be a DFA with s states accepting some subset of X^*. Then, for $g \in \{f(\text{mod } \mathcal{A}), f_{\mathcal{A}}\}$ we have*

1. $C(g) \leq C(f) \cdot (ns)^{O(1)}$.
2. $C^+(g) \leq C^+(f) \cdot (ns)^{O(1)}$.
3. $B(g) \leq B(f) \cdot (ns)^{O(1)}$.

Furthermore, the circuit (ABP) of the size given above for the polynomial g can be computed in deterministic logspace (hence in NC^2*) on input a circuit (resp. ABP) for f and the DFA $\mathcal{A}$.*

Proof We first describe a circuit construction that proves parts 1 and 2 of the theorem.

Let $\mathcal{A} = (Q, X, \delta, q_0, F)$ be the quintuple describing the given DFA with s states. We can extend the transition function δ to words (i.e. monomials) in X^* as usual: $\delta(a, m) = b$ for states $a, b \in Q$ and a monomial m if the DFA goes from state a to b on the monomial m. In particular, we note that $\delta(a, \epsilon) = a$ for each state a. As in automata theory, this is a useful convention because when we write a polynomial $f \in \mathbb{F}\{X\}$ as $\sum c_m m$, where c_m is the coefficient of the monomial m in f, we can allow for ϵ as the monomial corresponding to the constant term in f.

Let C be a circuit computing polynomial f. For each gate g of C, let f_g denote the polynomial computed by C at the gate g. In the new circuit C' we will have s^2 gates $\langle g, a, b\rangle, a, b \in Q$ corresponding to each gate g of C. Let $M_{ab} = \{m \in X^* \mid \delta(a, m) = b\}$ for states $a, b \in Q$. At the gate $\langle g, a, b\rangle$ the circuit C' will compute the polynomial $f_g^{a,b} = \sum_{m \in M_{ab} \cap mon(f_g)} c_m m$, where $f_g = \sum c_m m$.

The input-output connections between the gates of C' are now easy to define. If g is a $+$ gate with input gates h and k so that $f_g = f_h + f_k$, we have $f_g^{a,b} = f_h^{a,b} + f_k^{a,b}$, implying that $\langle h, a, b\rangle$ and $\langle k, a, b\rangle$ are the inputs to the $+$ gate $\langle g, a, b\rangle$. If g is a $\times$ gate, with inputs h and k so that $f_g = f_h \cdot f_k$, we have $f_g^{a,b} = \sum_{c\in Q} f_h^{a,c} \cdot f_k^{c,b}$.

This simple formula can be easily computed by a small subcircuit with $O(s)$ many $+$ gates and $\times$ gates. Finally, let out denote the output gate of circuit C, so that $f_{out} = f$. It follows from the definitions that $f(\text{mod } \mathcal{A}) = \sum_{a\notin F} f_{out}^{q_0,a}$ and $f_{\mathcal{A}} = \sum_{a\in F} f_{out}^{q_0,a}$.

Hence, by introducing a small formula for this computation with suitably designated output gate, we can easily get the circuit C' to compute $f(\text{mod } \mathcal{A})$ or $f_{\mathcal{A}}$.

To prove the correctness of the construction we need to show that, at any gate $\langle g, a, b\rangle$ for $a, b \in Q$ C' computes polynomial $f_g^{a,b} = \sum_{m_\alpha \in M_{ab} \cap \text{mon}(f_g)} c_\alpha m_\alpha$. It cab be shown easily using inductive argument.

Furthermore, $\text{size}(\text{C}')$ satisfies the claimed bound. Note that C' will remain a monotone circuit if the given circuit C is monotone. This completes the proof of the first two parts.

Now we prove part 3 of the statement. Let P be an ABP computing polynomial f and $\mathcal{A}$ be a given DFA. The idea for the construction of ABPs that compute $f(\text{mod } \mathcal{A})$ and $f_{\mathcal{A}}$ is quite similar to the construction described for part 1. Consider for instance the ABP P' for $f(\text{mod } \mathcal{A})$. Consider the directed acyclic layered graph underlying the ABP P. In the new ABP P' we will have exactly the same number of layers as for P. However, for each node b in the i^{th} layer of ABP P we will have nodes $\langle b, q\rangle$ for each state q of the DFA $\mathcal{A}$. Now, let f_b denote the polynomial that is computed at the node b by the ABP P. The property that the construction of P' can easily ensure is $f_b = \sum_q f_{\langle b,q\rangle}$, where $f_{\langle b,q\rangle}$ is the polynomial computed at node $\langle b, q\rangle$ by the ABP P'. More precisely, let M_q be the set of all nonzero monomials m of f_b such that on input m the DFA $\mathcal{A}$ goes from start state to state q. Then the polynomial $f_{\langle b,q\rangle}$ will be actually the sum of all the terms of the polynomial f_b corresponding to the monomials in M_q. This construction can now be easily used to obtain ABPs for each of $f(\text{mod } \mathcal{A})$ and $f_{\mathcal{A}}$, and the size of the ABP will satisfy the claimed bound. We omit the details of the construction.

A careful inspection of the constructions shows that for a given circuit C and DFA $\mathcal{A}$ we can construct the circuits that compute $C(\ \text{mod}\ \ \mathcal{A})$ and $C_{\mathcal{A}}$ in deterministic logspace

(and hence in NC^2). Likewise, the construction of the ABPs for $P(\mod \mathcal{A})$ and $P_{\mathcal{A}}$ for a given ABP P can also be computed in deterministic logspace. ∎

4.2.1 Identity Testing and Automata

Given an ABP computing a polynomial $f \in \mathbb{F}\{X\}$ and a DFA or an NFA $\mathcal{A}$ with s states, consider the question of checking whether the polynomial $f(\text{mod } \mathcal{A})$ is identically zero in $\mathbb{F}\{X\}$. If $\mathcal{A}$ is a deterministic finite automata using Theorem 4.5 and the identity testing algorithm of Raz and Shpilka [RS05] it immediately follows that the problem can be solved in deterministic polynomial time (in $size(B), s$).

In contrast, in the case of NFA's the problem turns out to be coNP-hard, even when the polynomial f is presented by formula instead of an ABP, more precisely:

Theorem 4.6 *Given a noncommutative formula F computing a polynomial $f \in \mathbb{Q}\{Z\}$ and an NFA A accepting language $L(A) \subseteq Z^*$ then the problem of testing whether the polynomial $f(\text{mod } \mathcal{A})$ is identically zero is* coNP*-hard.*

Proof We give a reduction from 3CNF-SAT to the complement of the problem. Let $S = C_1 \wedge C_2 \wedge \ldots \wedge C_t$ be a 3CNF formula where $C_i = c_{i1} \vee c_{i2} \vee c_{i3}$ for $1 \leq i \leq t$, and C_{ij}'s are from $\{w_1, \ldots, w_n\} \cup \{\neg w_1, \ldots, \neg w_n\}$. Let $f = \prod_{i=1}^{t} \sum_{j=1}^{3} z_{ij}$ where $z_{ij} = x_l$ if $c_{ij} = w_l$ and $z_{ij} = y_l$ if $c_{ij} = \neg w_l$ for $1 \leq l \leq n, 1 \leq j \leq 3$. Clearly, there is an $O(t)$ size formula F over indeterminates $Z = \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_n\}$ for the polynomial f. Let $L \subseteq Z^*$ be the set of all words of the form $m = ux_ivy_iw$ or $m = uy_ivx_iw$ for some $1 \leq i \leq n$. Clearly, there is an $O(n)$ size NFA $\mathcal{A}$ such that $L = L(\mathcal{A})$. Notice that the 3CNF formula S is satisfiable if and only if the polynomial $f(\text{mod } \mathcal{A})$ is *not* identically zero. Hence the given problem is coNP-hard. ∎

4.3 Noncommutative Monomial Algebras and Automata

In this section we prove our lower bound and identity testing results for noncommutative ABPs over monomial algebras using Theorem 4.5. First we recall some basic definitions.

A two-sided ideal I of the noncommutative polynomial ring $\mathbb{F}\{X\}$ is a subring of $\mathbb{F}\{X\}$ and which is closed under both right and left multiplication by the elements of $\mathbb{F}\{X\}$. Ideal I is specified by a generating set of polynomials. Here we are only concerned with the ideals generated by a finite set of polynomials. Ideal I is generated by a set of polynomials $S = \{f_1, \ldots, f_k\}$ if for every polynomial $f \in I$ there exist polynomial $g_1, \ldots, g_t, h_1, \ldots, h_t \in \mathbb{F}\{X\}$ and $f_{i_1}, \ldots, f_{i_t} \in S$ such that $f = \sum_{j=1}^{t} g_j f_{i_j} h_j$ and it is denoted by $I = \langle f_1, \ldots, f_k \rangle$.

Definition 4.7 *A two-sided ideal $I = \langle m_1, m_2, \cdots, m_r \rangle$ of the noncommutative ring $\mathbb{F}\{X\}$ generated by a finite set of monomials $m_1, \cdots, m_r$ is a* finitely generated monomial ideal *of $\mathbb{F}\{X\}$. The quotient algebra $\mathbb{F}\{X\}/I$ is a* finitely generated monomial algebra*.*

Note that polynomials in $\mathbb{F}\{X\}$ can be thought of as $\mathbb{F}$-linear combinations of words over alphabet X. It follows easily from definition that If I is a monomial ideal generated by $m_1, \ldots m_r$ then following is true: for every monomial m of a polynomial $f \in I$ there exists monomials $u, v \in X^*$ and $m_i \in I$ such that $m = um_iv$.

For a polynomial f given by an ABP and a monomial ideal I given by a generating set we are interested in the ABP complexity of the polynomial f over monomial algebra $\mathbb{F}\{X\}/I$. We denote ABP complexity of the polynomial f over $\mathbb{F}\{X\}/I$ by $B_I(f)$ and is equal to $\min_{g \in I} B(f + g)$. The corresponding identity testing problem is the *Ideal Membership* problem: whether the polynomial $f \in I$. First we prove our lower bound result.

Theorem 4.8 *Let $m_1, \cdots, m_r$ be monomials in the ring $\mathbb{F}\{x_{1,1}, \ldots, x_{n,n}\}$ and I be the monomial ideal generated by $m_1, \ldots, m_r$. Then $B_I(Perm_n) = 2^{\Omega(n)}$ if $r = 2^{o(n)}$.*

Proof To the contrary suppose that $B_I(Perm_n) = 2^{o(n)}$ so there exist an ABP B of size $2^{o(n)}$ computing homogeneous polynomial f of degree d such that $f(\mod I) = Perm_n$, i.e. there exist polynomial $g \in I$ and $f = Perm_n + g$.

Note that polynomials in $\mathbb{F}\{X\}$ can be thought of as $\mathbb{F}$-linear combinations of words over alphabet X. It follows easily from the definition that If I is a monomial ideal

generated by $m_1, \ldots m_r$ then following is true: for every monomial m of a polynomial $h \in I$ there exists monomials $u, v \in X^*$ and $m_i \in I$ such that $m = um_iv$. Since $Perm_n = (f \mod I)$, any monomial of $Perm_n$ can not have $m_1, m_2, \ldots, m_r$ as a substring and every monomial of g will contain one of the m_i's as a substring.

Without loss of generality we can assume that $d \geq \max_i\{\text{length}(m_i)\}$. Using the Aho-Corasick pattern matching automaton [AC75] we can construct a DFA A with $O(dr)$ states which on input a string $s \in X^*$ accepts s if s contains m_i as a substring for some i where $X = \{x_{1,1}, \ldots, x_{n,n}\}$. Clearly $L(\mathcal{A})$ will contain all the monomials of g and no monomial of $Perm_n$. So if we take quotient of B by a DFA $\mathcal{A}$ we will obtain Permanent polynomial. Using Theorem 4.5 we obtain an ABP B' of size $poly(n, d, r) = 2^{o(n)}$ which computes the polynomial $Perm_n = f(\text{mod } \mathcal{A})$, which is a contradiction due to Nisan's lower bound [N91]. ∎

It is easy to see that the lower bound result of Theorem 4.8 is also true for the Determinant polynomial (the same proof works). In fact the above lower bound result holds for any polynomial f which is weakly equivalent to the Permanent in the sense of Nisan [N91] (I.e. a monomial m occurs in mon(f) if and only if some shuffling of m occurs in mon$(Perm_n)$.)

Following theorem about Identity Testing problem for non commutative ABPs or circuits over monomial algebra also uses the same idea of pattern matching automaton and then invokes Theorem 4.5. Finally using well known identity testing algorithms for ABP ([RS05]) and polynomial degree circuits ([BW05]) the result follows.

Theorem 4.9 *Let $I = \langle m_1, \cdots, m_r \rangle$ be a monomial ideal in $\mathbb{F}\{X\}$. Let P (resp. C) be a noncommutative ABP (resp. a polynomial degree monotone circuit) computing a polynomial $f \in \mathbb{F}\{X\}$. Then there is a deterministic (resp. randomized) polynomial-time algorithm to test if the polynomial $f(\text{mod } \mathcal{I})$ is identically zero.*

4.4 Commutative monomial algebras

In this section we will prove a lower bound result for commutative monotone circuits over monomial algebras. We will consider the right kind of DFAs that capture commu-

tativity so that the constructions of Theorem 4.5 are meaningful and go through.

Definition 4.10 (Commutative Automata) *Let $w \in X^d$ be any string of length d over the alphabet $X = \{x_1, \cdots, x_n\}$. Let $w = w_1 w_2 \dots w_d$ where $w_i \in X$ for $i = 1$ to d. Let $C_w \subseteq X^d$ defined as follows:*

$$C_w = \bigcup_{\sigma \in S_d} w_{\sigma(1)} w_{\sigma(2)} \dots w_{\sigma(d)}$$

Where S_d is symmetric group on d elements. (i.e. C_w is the set of all words w' obtained by shuffling the letters of w.)

A DFA (or NFA) $\mathcal{A}$ over the alphabet $X = \{x_1, \cdots, x_n\}$ is said to be commutative *if for every word $w \in X^*$, w is accepted by $\mathcal{A}$ if and only if every word in C_w is accepted by $\mathcal{A}$.*

The following theorem is the analogue of Theorem 4.5 for intersecting and Quotienting commutative circuits and commutative ABPs by commutative DFAs. The constructions are identical to those in the proof of Theorem 4.5. It can be easily seen from Definition 4.4 and the proof of Theorem 4.5 that in the commutative case the constructions are meaningful and work correctly when the DFAs considered are commutative.

Theorem 4.11 *Let $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ and $\mathcal{A}$ be a commutative DFA with s states over alphabet $X = \{x_1, \cdots, x_n\}$. Then, for $g \in \{f(\text{mod } \mathcal{A}), f_{\mathcal{A}}\}$ we have*

1. $C(g) \leq C(f) \cdot (ns)^{O(1)}$.
2. $C^+(g) \leq C^+(f) \cdot (ns)^{O(1)}$.
3. $B(g) \leq B(f) \cdot (ns)^{O(1)}$.

Furthermore, the commutative circuit (ABP) for polynomial g meeting the above size bounds are computable in deterministic logspace (hence in NC^2*) on input a circuit (resp. ABP) for f and DFA $\mathcal{A}$.*

Now we consider an identity testing problem analogous to the one studied in Section 4.2.1. Given a monotone commutative circuit C computing a polynomial $f \in \mathbb{F}[X]$ and a commutative automaton $\mathcal{A}$ (DFA or NFA) our goal is to check whether $f(\text{mod } \mathcal{A})$ is identically zero or not. When $\mathcal{A}$ is a commutative DFA, it follows easily from Theorem 4.11 that the problem has a deterministic polynomial-time algorithm. In contrast, when $\mathcal{A}$ is a commutative NFA, we show that the problem is co-NP hard even when the polynomial f is given by a monotone formula instead of a monotone circuit.

Theorem 4.12 *Given a commutative monotone formula F computing a polynomial $f \in \mathbb{Q}[Z]$ and a commutative NFA A accepting language $L(A) \subseteq Z^*$ then the problem of testing whether the polynomial f(mod $\mathcal{A}$) is identically zero is* coNP*-hard.*

Proof of the Theorem 4.12 is similar to that of Theorem 4.6, just observe that the NFA defined in the proof of Theorem 4.6 is a commutative NFA.

Let $I = \langle m_1, \cdots, m_k \rangle$ be a monomial ideal contained in $\mathbb{F}[x_1, \cdots, x_n]$. As before $f(\mod I)$ is a meaningful polynomial in $\mathbb{F}[x_1, \cdots, x_n]$ for $f \in \mathbb{F}[x_1, \cdots, x_n]$.

We consider the problem for monotone circuits. It is useful to understand the connection between monotone noncommutative circuits and context-free grammars. For basics of language theory we refer to [HMU].

Definition 4.13 *We call a context-free grammar in Chomsky normal form $G = (V, T, P, S)$ an* acyclic CFG *if for any nonterminal $A \in V$ there does not exist any derivation of the form $A \Rightarrow^* uAw$.*

The size $size(G)$ of CFG $G = (V, T, P, S)$ in defined as the total number of symbols (in V, T, S) used in the production rules in P, where V, T, and P are the sets of variables, terminals, and production rules. It is clear that an acyclic CFG generates a finite language.We note the following easy proposition that relates acyclic CFGs to monotone noncommutative circuits over X.

Proposition 4.14 *For a monotone circuit C of size s computing a polynomial $f \in \mathbb{Q}\{X\}$ let mon(f) denote the set of nonzero monomials of f. Then there is an acyclic*

CFG G for mon(f) with $\mathrm{size(G)} = \mathrm{O(s)}$. *Conversely, if G is an acyclic CFG of size s computing some finite set $L \subset X^*$ of monomials over X, there exists a monotone circuit of size $O(s)$ that computes a polynomial $\sum_{m \in L} a_m m \in \mathbb{Q}\{X\}$, where the positive integer a_m is the number of derivation trees for m in the grammar G.*

Proof First we prove the forward direction by constructing an acyclic Context Free Grammar $G = (V, T, P, S)$ for mon(f). Let $V = \{A_g |$ g is a gate of circuit $C\}$ be the set of nonterminals of G. We include a production in P for each gate of the circuit C. If g is an input gate with input $x_i, 1 \leq i \leq n$ include the production $A_g \rightarrow x_i$ in P. If the input is a *nonzero* field element then add the production $A_g \rightarrow \epsilon$.[1] Let f_g denote the polynomial computed at gate g of C. If g is a $\times$ gate with $f_g = f_h \times f_k$ then include the production $A_g \rightarrow A_h A_k$ and if it is $+$ gate with $f_g = f_h + f_k$ include the productions $A_g \rightarrow A_h \mid A_k$. Let the start symbol $S = A_g$, where g is the output gate of C. It is easy to see from the above construction that G is acyclic moreover $size(G) = O(s)$ and it generates the finite language mon(f). The converse direction is similar. ∎

We need Lemma 4.15 to prove monotone circuit lower bound over commutative monomial algebras.

Lemma 4.15 *Let C be a monotone circuit computing a homogeneous polynomial $f \in \mathbb{Q}[x_1, x_2, \cdots, x_n]$ of degree d and let $\mathcal{A}$ be a commutative NFA of size s computing language $L(\mathcal{A}) \subseteq X^d$. There is a deterministic polynomial (in $size(C), s$) time algorithm to construct a monotone circuit C' which computes a polynomial $g \in \mathbb{Q}[x_1, x_2, \cdots, x_n]$ such that mon$(f_{\mathcal{A}})$ = mon(g).*

Proof For the given monotone circuit C we can consider it as a noncommutative circuit computing a polynomial f' in $\mathbb{Q}\{X\}$. Notice that f' is *weakly equivalent* to f in the sense of Nisan [N91]: I.e. a monomial m occurs in mon(f') if and only if some shuffling of m occurs in mon(f). Applying Proposition 4.14 we can obtain an acyclic context-free grammar G that generates precisely the finite set mon(f') of monomials of f'. The context-free grammar G is in Chomsky normal form. We can convert it into a

[1] If the circuit takes as input 0, we can first propagate it through the circuit and eliminate it.

Greibach normal form grammar G' in polynomial time, where the size of G' is $\mathrm{size(G)}^4$ [R67, KB97].

Now, we have a Greibach normal form grammar G' and NFA $\mathcal{A}$. We can apply the standard conversion of a Greibach normal form grammar to a pushdown automaton, to obtain a PDA M that accepts the same set that is generated by G'. The PDA M will encode each symbol of the grammar G' into binary strings of length $O(\log |G'|)$. Hence the PDA M will require an $O(\log |G'|)$ size worktape to simulate the transitions of the PDA apart from an unbounded stack. Now, it is easy to construct a new PDA M' that will simultaneously run the NFA $\mathcal{A}$ on the given input as well as the first PDA M so that M' accepts if and only if both the simulations accept. Clearly, M' will also require an $O(\log |G'|)$ size worktape to carry out this simulation. We can convert the PDA M' back into an acyclic grammar G'' in Chomsky normal form in polynomial time using a standard algorithm. From this acyclic CFG G'' we can obtain a monotone circuit C'', where the gates correspond to nonterminals. By construction it follows that C'' computes a polynomial f'' in $\mathbb{Q}\{X\}$ such that $\mathrm{mon}(f'') = \mathrm{mon}(f') \cap L(\mathcal{A})$. At this point we invoke the fact that $\mathcal{A}$ is a *commutative* NFA. Hence, we can view C'' as a commutative monotone circuit. Let $g \in \mathbb{Q}[x_1, \ldots, x_n]$ is a polynomial computed by C''. So it follows that $\mathrm{mon}(g) = \mathrm{mon}(f_{\mathcal{A}})$ which proves the lemma. ∎

Theorem 4.16 *Let $I = \langle m_1, \ldots, m_k \rangle$ be a commutative monomial ideal in $\mathbb{Q}[x_{11}, \ldots, x_{nn}]$, generated by $k = o(\frac{n}{\lg n})$ many monomials, such that $degree(m_i) \leq n^c$ for a constant c. Suppose C is a monotone circuit computing a polynomial f in $\mathbb{Q}[x_{11}, \ldots, x_{nn}]$ such that the permanent $Perm_n = f(\mod I)$ then $C^+(f) = 2^{\Omega(n)}$.*

Proof Let X denote the set of variables $\{x_{11}, \ldots, x_{nn}\}$. For each monomial m_t, $1 \leq t \leq k$ in the generating set for I write $m_t = \prod_{ij} x_{ij}^{e_{ijt}}$, where e_{ijt} are nonnegative integers for $1 \leq i, j \leq n, 1 \leq t \leq k$.

Consider the language $L \subset X^*$ containing all strings m such that for each $t, 1 \leq t \leq k$ there exist $i, j \in [n]$ such that the number of occurrences of x_{ij} in m is strictly less that e_{ijt}. Notice that L is precisely $X^* \setminus I$. Clearly, the language L is *commutative*: if $m \in L$ then so is every reordering of the word m. It is easy to see that there is a *commutative* NFA $\mathcal{A}$ with $n^{O(k)} = 2^{o(n)}$ states such that $L = L(\mathcal{A})$ (the NFA is designed using

counters for each t and guessed i, j, note that $e_{ijt} \leq n^c$, for $1 \leq t \leq k, 1 \leq i, j \leq n$). So we have $Perm_n = f_{\mathcal{A}}$.

Suppose the polynomial f can be computed by a monotone circuit C of size $2^{o(n)}$. By Lemma 4.15 there is a monotone circuit of size $2^{o(n)}$ computing a polynomial g such that $\text{mon}(g) = \text{mon}(f_{\mathcal{A}}) = \text{mon}(Perm_n)$. We observe that the $2^{\Omega(n)}$ size lower bound proof for commutative circuits computing the permanent (specifically, the Jerrum-Snir work [JS82]) also imply the same lower bound for the polynomial g, because the coefficients do not play a role and $\text{mon}(g) = \text{mon}(Perm_n)$. This completes the proof. ∎

Corollary 4.17 *Let C be a commutative monotone arithmetic circuit computing polynomial $f \in \mathbb{Q}[x_1, \ldots, x_n]$ and let $I = \langle m_1, \ldots, m_k \rangle$ be a commutative monomial ideal generated by $k = o(n/\log n)$ monomials, such that for $1 \leq t \leq k$, $degree(m_t) \leq n^c$ for a constant c. Then the problem of testing whether $f \in I$ can be solved in deterministic $2^{o(n)} \cdot poly(size(C))$ time.*

Proof Let $X = \{x_1, \ldots, x_n\}$. As in the proof of Theorem 4.16 we can construct an NFA $\mathcal{A}$ of size $2^{o(n)}$ such that $L(\mathcal{A}) = X^* \setminus I$. By Lemma 4.15 we can construct a monotone commutative circuit C' of size $2^{o(n)} \cdot poly(size(C))$ computing polynomial g such that $mon(g) = mon(f_{\mathcal{A}})$. It is clear that $f \in I$ iff $f_{\mathcal{A}}$ is identically zero iff g is identically zero. We can test if g is identically zero using standard algorithms. ∎

Given an ABP (or monotone circuit) of size s computing some polynomial f in the noncommutative ring $\mathbb{Q}\{x_1, \ldots, x_n\}$ and a noncommutative monomial ideal $I = \langle m_1, \ldots, m_k \rangle$ we can test if $f \in I$ in deterministic time $2^{o(n)} s^{O(1)}$ even when the number of monomials k generating I is $k = 2^{o(n)}$. On the other hand, in the commutative setting we are able to show a similar result (Corollary 4.17) only for $k = o(n/\log n)$. Nevertheless, it appears difficult to prove a significantly stronger result. We can show that strengthening Corollary 4.17 to $k = \frac{n}{2}$ would imply that 3CNF-SAT has a $2^{o(n)}$ time algorithm contradicting the exponential-time hypothesis [IPZ01]. We make this statement precise in the next result (whose proof is similar to that of Theorem 4.6).

Theorem 4.18 *Given a commutative monotone circuit C of size s computing a polynomial $f \in \mathbb{Q}[Z]$ in $2n$ variables and a commutative monomial ideal $I = \langle m_1, \ldots, m_k \rangle$,*

$k \geq n$ then the problem of testing if $f \in I$ is coNP*-hard. Specifically, for $k = n$ the problem of testing if $f \in I$ does not have a $2^{o(n)}s^{O(1)}$ time algorithm assuming the exponential-time hypothesis.*

Proof We give a reduction from 3CNF-SAT to the complement of the problem. Let $S = C_1 \wedge C_2 \wedge \ldots \wedge C_t$ be a 3CNF formula where $C_i = c_{i1} \vee c_{i2} \vee c_{i3}$ for $1 \leq i \leq t$, and C_{ij}'s are from $\{w_1, \ldots, w_n\} \cup \{\neg w_1, \ldots, \neg w_n\}$. Let $f = \prod_{i=1}^{t} \sum_{j=1}^{3} z_{ij}$ where $z_{ij} = x_i$ if $c_{ij} = w_i$ and $z_{ij} = y_i$ if $c_{ij} = \neg w_i$ for $1 \leq i \leq n, 1 \leq j \leq 3$. Clearly, there is an $O(t)$ size monotone circuit (in fact formula) C over indeterminates $Z = \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_n\}$ for the polynomial f. Let I be a monomial ideal generated monomials $x_1y_1, x_2y_2, \ldots x_ny_n$. It is easy to see that formula S is not satisfiable iff $f \in I$. ∎

In contrast to Theorem 4.16, we observe that $Perm_n$ can be computed by a small monotone formula modulo a monomial ideal generated by $O(n^3)$ many monomials.

Theorem 4.19 *There is a monomial ideal $I = \langle m_1, \cdots, m_t \rangle$ of $\mathbb{F}[X]$, where $X = \{x_{ij} \mid 1 \leq i, j \leq n\}$,$t = O(n^3)$ and a polynomial-sized commutative monotone formula $F(x_{11}, \cdots, x_{nn})$ such that $Perm_n = F(\mod I)$.*

Proof Let $F = \prod_{i=1}^{n}(x_{i1} + x_{i2} + \cdots + x_{in})$ and I be the monomial ideal generated by the set of monomials $\{x_{ik}x_{jk} \mid 1 \leq i, j, k \leq n\}$. Clearly, $Perm_n = F(\mod I)$. ∎

4.5 Monomial search problem

We now consider the monomial search problem for ABPs in both commutative and noncommutative setting. The problem is to find a nonzero monomial of the polynomial computed by a given ABP. We show that in both the cases the problem is in randomized NC^2. The basic idea of our proofs is to assign random weights to the indeterminates x_i's, from isolation lemma we can argue that there is a unique monomial of f (a polynomial computed by given ABP) with minimum weight. We can construct a *weight-checking*

DFA which accepts a monomial of a particular weight. Then using theorem on intersection on ABPs by DFA we recover a nonzero monomial in f. Our results builds on ideas from [AM08, AMS08]. First we consider the noncommutative case.

Theorem 4.20 *Given a noncommutative ABP P computing a polynomial f in $\mathbb{F}\{X\}$ there is a randomized* NC^2 *algorithm that computes a nonzero monomial of f. More precisely, the algorithm is a randomized* $\mathrm{FL}^{\mathrm{GapL}}$ *algorithm.*

Proof We can assume wlog that the given ABP P computes a homogeneous degree d polynomial. The proof is by an application of the isolation lemma of [**?**]. Define the universe $U = \{x_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq d\}$, where the element x_{ij} stands for the occurrence of x_i in the j^{th} position in a monomial. With this encoding every degree d monomial m over X can be encoded as a subset S_m of size d in U, where $S_m = \{x_{ij} \mid x_i \text{ occurs in } j^{th} \text{ position in } m\}$. Following the isolation lemma, we pick a random weight assignment $w : U \longrightarrow [4dn]$. The weight of a monomial m is defined as $w(m) = w(S_m) = \sum_{x_{ij} \in S_m} w(x_{ij})$, and with probability $1/2$ there is a unique minimum weight monomial.

Construction of weight-checking DFA: For any weight value a such that $1 \leq a \leq 4nd^2$, we can easily construct a DFA M_w^a that accepts a monomial $m \in X^*$ iff $m \in X^d$ and $w(m) = a$. This DFA will have $O(4nd^3)$ many states. Furthermore, we can compute this DFA in deterministic logspace. Next, by Theorem 4.5 we can compute an ABP P_w^a that computes the polynomial $P(\text{div } M_w^a)$ for each of $1 \leq a \leq 4nd^3$. With probability $1/2$ we know that one of $P(\text{div } M_w^a)$ accepts precisely one monomial of the original polynomial f (with the same coefficient).

In order to find each variable occurring in that unique monomial accepted by, say, P_w^a we will design another DFA $\mathcal{A}_{ij}$ which will accept a monomial $m \in X^d$ if and only if x_i occurs in the j^{th} position. Again by Theorem 4.5 we can compute an ABP $B_{i,j,a,w}$ that accepts precisely $P_w^a(\text{div } \mathcal{A}_{ij})$. Now, the ABP $B_{i,j,a,w}$ either computes the zero polynomial (if x_i does not occur in the j^{th} position of the unique monomial of P_w^a) or it computes that unique monomial of P_w^a. In order to test which is the case, notice that we can *deterministically* assign the values $x_i = 1$ for each variable x_i. Crucially, since P_w^a has a *unique* monomial it will be nonzero even for this deterministic and commutative

evaluation. Since the evaluation of an ABP is for commuting values (scalar values), we can carry it out in NC^2 in fact, in FL^{GapL} for any fixed finite field or over $\mathbb{Q}$, (see e.g. [T91], [V91], [MV97]).

Let m be the monomial that is finally constructed. We can construct a DFA $\mathcal{A}_m$ that accepts only m and no other strings. By Theorem 4.5 we can compute an ABP P' for the polynomial $P(\text{div } \mathcal{A}_m)$ and again check if P' is zero or nonzero by substituting all $x_i = 1$ and evaluating. This will make the algorithm actually a zero-error NC^2 algorithm. The success probability can be boosted by parallel repetition. ∎

Next we describe a randomized NC^2 algorithm for the Monomial search problem for commutative ABPs. This is the best we can currently hope for, since deterministic polynomial-time identity testing for commutative ABPs is a major open problem. Our monomial search algorithm is based on a generalized isolation lemma [KS01].

Lemma 4.21 [KS01, Lemma 4] *Let L be a collection of linear forms over variables $z_1, \ldots, z_n$ with integer coefficients in $\{0, 1, \ldots, K\}$. If each z_i is picked independently and uniformly at random from $\{0, 1, \ldots, 2Kn\}$ then with probability at least $\frac{1}{2}$ there is a unique linear form in L which attains minimum value at $(z_1, z_2, \ldots, z_n)$.*

Theorem 4.22 *The monomial search problem for commutative algebraic branching programs is in randomized* NC^2 *(more precisely, it is in randomized* FL^{GapL}*).*

Proof Let P be a commutative algebraic branching program computing a polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$. We can assume, without loss of generality, that f is homogeneous of degree d. First, pick a random weight function $w : \{x_1, \cdots, x_n\} \longrightarrow [2dn]$. Next, for each number a such that $0 \leq a \leq 2d^2n$ we construct a DFA A_w^a which will accept a monomial $m \in X^*$ iff $m \in X^d$ and $w(m) = a$, where $w(m) = \sum_i w(x_i) \cdot \alpha_i$, and x_i occurs exactly α_i times in m. Crucially, notice that A_w^a is a *commutative* DFA. Hence, applying Theorem 4.11, for each number a we can obtain an ABP P_w^a in deterministic logspace.

By Lemma 4.21 with probability at least $1/2$ one of the ABPs P_w^a accepts a unique monomial $m = x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n}$ of f. Suppose that value of a is u. Let $c \neq 0$ denote

the coefficient of the unique monomial m in f computed by the ABP P_w^u. We need to compute each α_i. We evaluate the ABP P_w^u by setting $x_j = 1$ for all $j \neq i$ to obtain $cx_i^{\alpha_i}$. Evaluating the ABPs P_w^a, for each a, on the inputs $(1, \cdots, 1, x_i, 1, \cdots, 1)$ can be done in NC^2. Indeed, it can be done in FL^{GapL}, since we only need determinant computation over the field $\mathbb{F}$. This completes the proof sketch. ∎

Theorem 4.23 *There is a deterministic polynomial time algorithm for the monomial search problem for noncommutative algebraic branching programs.*

Proof W.l.o.g. assume that the input noncommutative ABP P computes a degree d homogeneous polynomial $f = \sum_m a_m m$. The monomial search algorithm is a simple prefix search guided by the Raz-Shpilka deterministic identity test [RS05]. Starting with $w = \epsilon$, we successively compute ABPs $P_\epsilon, P_{w_1}, \cdots, P_{w_d}$, where $|w_k| = k$ and w_k is a prefix of w_{k+1} for each k. Each P_{w_k} is an ABP that computes $f(\text{div } D_{w_k})$ where D_{w_k} is a DFA that accepts all the words with prefix w. The prefix search sets $w_{k+1} = w_k x_i$ for the first indeterminate x_i such that $P_{w_{k+1}}$ computes a nonzero polynomial (to check this we use the Raz-Shpilka identity test on $P_{w_{k+1}}$ [RS05]). Since $f(\text{div } D_{w_k}) \neq 0$ for some indeterminate x_i the polynomial $f(\text{div } D_{w_{k+1}})$ is nonzero. Hence the prefix search will successfully continue. The output of the monomial search will be w_d. ∎

Finally, our technique of isolating a monomial using DFAs along with intersecting circuits with DFAs can be applied to get a randomized NC reduction from monomial search for noncommutative (or commutative) circuits to noncommutative (resp. commutative) polynomial identity testing.

Theorem 4.24 *Monomial search for noncommutative (commutative) circuits is randomized* NC *reducible to noncommutative (resp. commutative) polynomial identity testing.*

4.6 Overview

For an ideal I of the noncommutative (resp. commutative) polynomial ring $\mathbb{F}\{x_1, \ldots, x_n\}$ (resp. $\mathbb{F}[x_1, \ldots, x_n]$) we explored the lower bound questions over the quotient algebras

$\mathbb{F}\{x_1, \ldots, x_n\}/I$ (resp. $\mathbb{F}[x_1, \ldots, x_n]/I$. When ideal I is generated by monomials in the input variables $x_1, \ldots, x_n$ such a quotient algebra is called as monomial algebra. Using automata theoretic techniques we obtained following results:

Let $m_1, \cdots, m_r$ be monomials over noncommuting indeterminates $x_{1,1}, \ldots, x_{n,n}$ and I be a monomial ideal of $\mathbb{F}\{x_{1,1}, \ldots, x_{n,n}\}$ generated by $m_1, \ldots, m_r$. Then if $r = 2^{o(n)}$ the branching program complexity of the Permanent over $\mathbb{F}\{x_{1,1}, \ldots, x_{n,n}\}/I$ is $2^{\Omega(n)}$. This basically extends Nisan's exponential size lower bound result [N91] for noncommutative algebraic branching program over free noncommutative polynomial ring to a similar lower bound result for noncommutative ABP's over noncommutative monomial algebras.

In commutative setting we showed a weaker result. If $I = \langle m_1, \ldots, m_k \rangle$ is a commutative monomial ideal in $\mathbb{Q}[x_{11}, \ldots, x_{nn}]$, generated by $k = o(\frac{n}{\lg n})$ many monomials, such that $degree(m_i) \leq n^c$ for a constant c then we showed that the monotone circuit complexity of the Permanent over the monomial algebra $\mathbb{Q}[x_{11}, \ldots, x_{nn}]/I$ is $2^{\Omega(n)}$. This extends Jerrum and Snir's lower bound result [JS82] to analogous lower bound result over monomial algebras.

We also explored monomial search problem, which is a natural search version of the identity testing problem. When the input polynomial is given by an ABP, we showed a randomized NC^2 upper bound on the complexity of the problem for both commutative and noncommutative setting. In noncommutative setting we showed deterministic polynomial time upper bound on the complexity of the problem.

5

Hadamard Product of Polynomials and the Identity Testing Problem

In this chapter we introduce and study the Hadamard product of the multivariate polynomials in the free noncommutative polynomial ring $\mathbb{F}\{x_1, x_2, \cdots, x_n\}$. We explore arithmetic circuit and branching program complexity of the Hadamard product of polynomials when they are individually given by arithmetic circuits and/or algebraic branching programs.

5.1 Introduction

Our definition of the Hadamard Product can be seen as an algebraic generalization of the intersection of the formal languages. The definition of Hadamard Product is motivated by the well known Hadamard product of matrices. Hadamard product of matrices of same dimension is simply entry-wise product. Next we define the Hadamard product of polynomials.

Definition 5.1 *Let $f, g \in \mathbb{F}\{X\}$ where $X = \{x_1, x_2, \cdots, x_n\}$. The* Hadamard product *of f and g, denoted $f \circ g$, is the polynomial $f \circ g = \sum_m a_m b_m m$, where $f = \sum_m a_m m$ and $g = \sum_m b_m m$, where the sums index over monomials m.*

To see the connection of this definition with that of Hadamard product of two matrices we recall the definition of communication matrices [N91] associated with a degree d homogeneous polynomial $f \in \mathbb{F}\{X\}$. For $k = 1, \ldots, d$ the communication matrix $M_k(f)$ has its rows indexed by degree k monomials and columns by degree $d - k$ monomials and the $(m, m')^{th}$ entry of $M_k(f)$ is the coefficient of mm' in f. It follows easily that Hadamard product of communication matrices associated with two polynomials f and g is same as the communication matrix associated with their Hadamard product (as defined above).

We show that the *noncommutative* branching program complexity of the Hadamard product $f \circ g$ is upper bounded by the product of the branching program sizes for f and g. This upper bound is natural because we know from Nisan's seminal work [N91] that the algebraic branching program (ABP) complexity $B(f)$ is well characterized by the ranks of its "communication" matrices $M_k(f)$, and the rank of Hadamard product $A \circ B$ of two matrices A and B is upper bounded by the product of their ranks. Our proof is constructive: we give a deterministic logspace algorithm for computing an ABP for $f \circ g$.

We then apply this result to tightly classify the identity testing problem for noncommutative ABPs over field of rationals. Before stating our main result we recall some complexity theory preliminaries.

We recall some definitions of logspace counting classes from [AO96]. Let L denote the class of languages accepted by deterministic logspace machines.

GapL is the class of functions $f : \Sigma^* \rightarrow \mathbb{Z}$, for which there is a logspace bounded NDTM M such that for each input $x \in \Sigma^*$, we have $f(x) = \mathit{acc}_M(x) - \mathit{rej}_M(x)$, where $\mathit{acc}_M(x)$ and $\mathit{rej}_M(x)$ are the number of accepting and rejecting paths of M on input x, respectively.

A language L is in $\mathrm{C_{=}L}$ if there exists a function $f \in$ GapL such that $x \in L$ if and only if $f(x) = 0$. For a prime p, a language L is in the complexity class $\mathrm{Mod}_p\mathrm{L}$ if there exists a function $f \in$ GapL such that $x \in L$ if and only if $f(x) = 0(\mod p)$.

It is shown in [AO96] that checking if an integer matrix is singular is complete for $\mathrm{C_{=}L}$ with respect to logspace many-one reductions. The same problem is known to be

complete for Mod_pL over a field of characteristic p. It is useful to recall that both $\text{C}_=\text{L}$ and Mod_pL are contained in NC^2.

Main results in this chapter:

It is shown by Raz and Shpilka [RS05] that the polynomial identity testing problem for noncommutative ABPs can be solved in deterministic polynomial time. Using result on Hadamard Product of two ABPs, we show that the identity testing problem for non-commutative ABPs over *rationals* is equivalent to the matrix singularity problem under deterministic logspace many-one reductions. This implies the identity testing problem in case of rationals is $\text{C}_=\text{L}$-complete, In particular it gives NC^2 upper bound on the complexity of the problem.

We show that, the identity testing problem for the noncommutative ABPs over finite field of characteristic p is equivalent to Matrix Singularity problem over field of characteristic p under randomized logspace reduction. Firstly this reduction shows a randomized NC^2 upper bound on the complexity of the problem and it follows from [AO96] that the problem is in randomized Mod_pL. Using standard amplification techniques we get a $\text{Mod}_p\text{L}/\text{Poly}$ upper bound. We also investigate the parallel complexity of the problem. We show that Raz-Shpilka identity test can be parallelized which gives a NC^3 upper bound for the identity testing problem for the non-commutative ABPs over any field.

It turns out that the problem is hard (with respect to logspace many-one reductions) for both NL and Mod_pL. Hence, it is not likely to be easy to improve the upper bound unconditionally to Mod_pL (it would imply that NL is contained in Mod_pL). However, under a hardness assumption we can apply standard arguments [ARZ99, KvM02] to derandomize this algorithm and put the problem in Mod_pL.

It is an interesting question whether we can show deterministic NC^2 upper bound on the identity testing problem for noncommutative ABPs over finite fields.

Next explore the expressive power of the Hadamard product of two polynomials when either or both of them given by arithmetic circuit. We show that if either of the two polynomials is given by an ABP the we can efficiently (in logspace) compute an arithmetic circuit for the Hadamard product of the polynomials. But if both the polynomials are give by arithmetic circuits then it is not easy to come up with an efficient algorithm

to compute an arithmetic circuit for the Hadamard product of the two polynomials (We show that such an algorithm would imply a non-trivial circuit-size lower bound).

We also consider following identity testing question: Given two polynomials $f, g \in \mathbb{F}\{X\}$ either by an ABP or by an arithmetic circuit check whether $f \circ g$ is identically zero. We show that if both the polynomials are given by arithmetic circuits then the problem is coNP-hard even when the circuits are monotone. Whereas, if either of the polynomials is given by an ABP the problem has polynomial time algorithm.

5.2 The Hadamard Product

Let $f, g \in \mathbb{F}\{X\}$ where $X = \{x_1, x_2, \cdots, x_n\}$. Clearly, mon$(f \circ g) =$ mon$(f) \cap$mon(g). Thus, the Hadamard product can be seen as an algebraic version of the intersection of formal languages. Our definition of the Hadamard product of polynomials is actually motivated by the well-known Hadamard product $A \circ B$ of two $m \times n$ matrices A and B. We recall the following well-known bound for the rank of the Hadamard product.

Proposition 5.2 *Let A and B be $m \times n$ matrices over a field $\mathbb{F}$. Then* rank$(A \circ B) \leq$ rank(A)rank(B).

It is known from Nisan's work [N91] that the ABP complexity $B(f)$ of a polynomial $f \in \mathbb{F}\{X\}$ is closely connected with the ranks of the communication matrices $M_k(f)$, where $M_k(f)$ has its rows indexed by degree k monomials and columns by degree $d-k$ monomials and the $(m, m')^{th}$ entry of $M_k(f)$ is the coefficient of mm' in f. Nisan showed that $B(f) = \sum_k$ rank$(M_k(f))$. Nisan's result and the above proposition easily imply the following bound on the ABP complexity of $f \circ g$.

Lemma 5.3 *For $f, g \in \mathbb{F}\{X\}$ we have $B(f \circ g) \leq B(f)B(g)$.*

Proof By Nisan's result $B(f \circ g) = \sum_k$ rank$(M_k(f \circ g))$. The above proposition implies

$$\begin{aligned}\sum_k \text{rank}(M_k(f \circ g)) &\leq \sum_k \text{rank}(M_k(f))\text{rank}(M_k(g)) \\ &\leq (\sum_k \text{rank}(M_k(f))(\sum_k \text{rank}(M_k(g)))\end{aligned}$$

and the claim follows. ∎

We now show an algorithmic version of this upper bound.

Theorem 5.4 *Let P and Q be two given ABP's computing polynomials f and g in $\mathbb{F}\{x_1, x_2, \ldots, x_n\}$, respectively. Then there is a deterministic polynomial-time algorithm that will output an ABP R for the polynomial $f \circ g$ such that the size of R is a constant multiple of the product of the sizes of P and Q. (Indeed, R can be computed in deterministic logspace.)*

Proof Let f_i and g_i denote the i^{th} homogeneous parts of f and g respectively. Then $f = \sum_{i=0}^{d} f_i$ and $g = \sum_{i=0}^{d} g_i$. Since the Hadamard product is distributive over addition and $f_i \circ g_j = 0$ for $i \neq j$ we have $f \circ g = \sum_{i=0}^{d} f_i \circ g_i$. Thus, we can assume that both P and Q are homogeneous ABP's of degree d. Otherwise, we can easily construct an ABP to compute $f_i \circ g_i$ separately for each i and put them together. Note that we can easily compute ABPs for f_i and g_i in logspace given as an input the ABPs for f and g.

By allowing parallel edges between nodes of P and Q we can assume that the labels associated with each edge in an ABP is either 0 or αx_i for some variable x_i and scalar $\alpha \in \mathbb{F}$. Let s_1 and s_2 bound the number of nodes in each layer of P and Q respectively. Denote the j^{th} node in layer i by $\langle i, j \rangle$ for ABPs P and Q. Now we describe the construction of the ABP R for computing the polynomial $f \circ g$. Each layer i, $1 \leq i \leq d$ of R will have $s_1 \cdot s_2$ nodes, with node labeled $\langle i, a, b \rangle$ corresponding to the node $\langle i, a \rangle$ of P and the node $\langle i, b \rangle$ of Q. We can assume that there is an edge from the every node in the layer i to the every node in the layer $i + 1$ for both ABPs. If there is no such edge we can always include it with label 0.

In the new ABP R we put an edge from $\langle i, a, b \rangle$ to $\langle i + 1, c, e \rangle$ with label $\alpha\beta x_t$ if and only if there is an edge from node $\langle i, a \rangle$ to $\langle i + 1, c \rangle$ with label αx_t in P and an edge from

$\langle i, b\rangle$ to $\langle i+1, e\rangle$ with label βx_t in ABP Q. Let $\langle 0, a, b\rangle$ and $\langle d, c, e\rangle$ denote the source and the sink nodes of ABP R, where $\langle 0, a\rangle, \langle 0, b\rangle$ are the source nodes of P and Q, and $\langle d, c\rangle, \langle d, e\rangle$ are the sink nodes of P and Q respectively. It is easy to see that ABP R can be computed in deterministic logspace. Let $h_{\langle i,a,b\rangle}$ denote the polynomial computed at node $\langle i, a, b\rangle$ of ABP R. Similarly, let $f_{\langle i,a\rangle}$ and $g_{\langle i,b\rangle}$ denote the polynomials computed at node $\langle i, a\rangle$ of P and node $\langle i, b\rangle$ of Q. We can easily check that $h_{\langle i,a,b\rangle} = f_{\langle i,a\rangle} \circ g_{\langle i,b\rangle}$ by an induction argument on the number of layers in the ABPs. It follows from this inductive argument that the ABP R computes the polynomial $f \circ g$ at its sink node. The bound on the size of R also follows easily. ∎

5.3 Identity Testing for noncommutative ABPs

In this section we explore the complexity of the polynomial identity testing problem for noncommutative ABPs. First we show that applying the above theorem we can get a *tight* complexity theoretic upper bound for the identity testing problem for noncommutative ABPs over rationals.

Theorem 5.5 *The problem of polynomial identity testing for noncommutative algebraic branching programs over $\mathbb{Q}$ is in* NC^2. *More precisely, it complete for the logspace counting class* $\mathrm{C_{=}L}$ *under logspace reductions.*

Proof Let P be the given ABP computing $f \in \mathbb{Q}\{X\}$. We apply the construction of the Theorem 5.4 to compute a polynomial sized ABP R for the Hadamard product $f \circ f$. Notice that $f \circ f$ is nonzero iff f is nonzero. Now, we crucially use the fact that $f \circ f$ is a polynomial whose nonzero coefficients are all *positive*. Hence, $f \circ f$ is nonzero iff it evaluates to a nonzero number on the all 1's input. The problem thus boils down to checking if R evaluates to a nonzero number on the all 1's input.

By Theorem 5.4, the ABP R for polynomial $f \circ f$ is computable in deterministic logspace, given as input an ABP for f. Furthermore, evaluating the ABP R on the all 1's input can be easily converted to iterated integer matrix multiplication (one matrix for each layer of the ABP), and checking if R evaluates to a nonzero number can be done by checking if a specific entry of the product matrix is nonzero. It is well known

that checking if a specific entry of an iterated integer matrix product is zero is in the logspace counting class $\mathrm{C_=L}$ (e.g. see [AO96, ABO99]). However, $\mathrm{C_=L}$ is contained in NC^2.

We now argue that the problem is hard for $\mathrm{C_=L}$. The problem of checking if an integer matrix A is singular is well known to be complete for $\mathrm{C_=L}$ under deterministic logspace reductions. The standard GapL algorithm for computing $\det(A)$ [T91] can be converted to an ABP P_A which will compute $\det(A)$. Hence the ABP P_A computes the identically zero polynomial iff A is singular. Putting it all together, it follows that identity testing of noncommutative ABPs over rationals is complete for the class $\mathrm{C_=L}$. ∎

An iterative matrix product problem Suppose B is a noncommutative ABP computing a homogeneous polynomial in $\mathbb{F}\{X\}$ of degree d, where each edge of the ABP is labeled by a linear form in variables from X.

Let n_ℓ denote the number of nodes of B in layer ℓ, $0 \leq \ell \leq d$. For each x_i and layer ℓ, we associate an $n_\ell \times n_{\ell+1}$ matrix $A_{i,\ell}$ where the $(k,j)^{th}$ entry of matrix $A_{i,\ell}$ is the coefficient of x_i in the linear form associated with the (v_k, u_j) edge in the ABP B. Here v_k is the k^{th} node in layer ℓ and u_j the j^{th} node in the layer $\ell+1$. The following claim is easy to see and relates these matrices to the ABP B.

Claim 8 *The coefficient of any degree d monomial $x_{i_1}x_{i_2}\cdots x_{i_d}$ in the polynomial computed by the ABP B is the matrix product $A_{i_1,0}A_{i_2,1}\cdots A_{i_d,d-1}$ (which is a scalar since $A_{i_1,0}$ is a row and $A_{i_d,d-1}$ is a column).*

Let i and j be any two nodes in the ABP B. We denote by $B(i,j)$ the algebraic branching program obtained from the ABP B by designating node i in B as the source node and node j as the sink node. Clearly, $B(i,j)$ computes a homogeneous polynomial of degree $b-a$ if i appears in layer a and j in layer b.

For layers a, b, $0 \leq a < b \leq d$ let $t = b-a$ and $P(a,b) = \{A_{s_1,a}A_{s_2,a+1}\ldots A_{s_t,b-1} | 1 \leq s_j \leq n, \text{ for } 1 \leq j \leq t\}$. $P(a,b)$ consists of $n_a \times n_b$ matrices. Thus the dimension of the linear space spanned by $P(a,b)$ is bounded by $n_a n_b$. It follows from Claim 8 that the linear span of $P(a,b)$ is the zero space iff the polynomial computed by ABP $B(i,j)$ is identically zero for every $1 \leq i \leq n_a$ and $1 \leq j \leq n_b$.

Thus, it suffices to compute a basis for the space spanned by matrices in $P(0,d)$ to check whether the polynomial computed by B is identically zero. We can easily give a deterministic NC^3 algorithm for this problem over any field $\mathbb{F}$: First recursively compute bases M_1 and M_2 for the space spanned by matrices in $P(0,d/2)$ and $P(d/2+1,d)$ respectively. From bases M_1 and M_2 we can compute in deterministic NC^2 a basis M for space spanned by matrices in $P(0,d)$ as follows. We compute the set S of pairwise products of matrices in M_1 and M_2 and then we can compute a maximal linearly independent subset of S in NC^2 (see e.g. [ABO99]). This gives an easy NC^3 algorithm to compute a basis for the linear span of $P(0,d)$. This proves the following.

Theorem 5.6 *The problem of polynomial identity testing for noncommutative algebraic branching programs over any field (in particular, finite fields $\mathbb{F}$) is in deterministic* NC^3.

Can we give a tight complexity characterization for identity testing of noncommutative ABPs over finite fields? We show that the problem is in nonuniform Mod$_p$L and is hard for Mod$_p$L under logspace reductions. Furthermore, the problem is hard for NL. Hence, it appears difficult to improve the upper bound to uniform Mod$_p$L (as NL is not known to be contained in uniform Mod$_p$L).

Theorem 5.7 *The problem of polynomial identity testing for noncommutative algebraic branching programs over a finite field $\mathbb{F}$ of characteristic p is in* Mod$_p$L/Poly.

Proof Consider a new ABP B' in which we replace the variables x_i, $1 \leq i \leq n$ appearing in the linear form associated with an edge from some node in the layer l to a node in the layer $l+1$ of ABP B by new variable $x_{i,l}$, for layers $l = 0, 1, \ldots, d-1$. Let $g \in \mathbb{F}[X]$ denotes the polynomial computed by the ABP B' in *commuting* variables $x_{i,l}, 1 \leq i \leq n, 1 \leq l < d$. It is easy to see that the commutative polynomial $g \in \mathbb{F}[X]$ is identically zero iff the noncommutative polynomial $f \in \mathbb{F}\{X\}$ computed by ABP B is identically zero. Now, we can apply the standard Schwartz-Zippel lemma to check if g is identically zero by substituting random values for the variables $x_{i,l}$ from $\mathbb{F}$ (or a suitable finite extension of $\mathbb{F}$). After substitution of field elements, we are left with an iterated matrix product over a field of characteristic p which can be done in Mod$_p$L. This gives us a randomized Mod$_p$L algorithm. By standard amplification it follows that the problem is in Mod$_p$L/Poly. ∎

Next we show that identity testing problem for noncommutative ABPs over any field is hard for NL by a reduction from directed graph reachability. Let (G, s, t) be a reachability instance. Without loss of generality, we assume that G is a layered directed acyclic graph. The graph G defines an ABP with source s and sink t as follows: label each edge e in G with a *distinct* variable x_e and for each absent edge put the label 0. The polynomial computed by the ABP is nonzero if and only if there is a directed s-t path in G.

Theorem 5.8 *The problem of polynomial identity testing for noncommutative algebraic branching programs over any field is hard for* NL.

5.4 Hadamard product of noncommutative circuits

In this section we study the expressive power of Hadamard product of two polynomial when one or both of them are given by an arithmetic circuits rather than an ABP. Analogous to Theorem 5.4 we show that $f \circ g$ has small circuits if f has a small circuit and g has a small ABP.

Theorem 5.9 *Let $f, h \in \mathbb{F}\{x_1, x_2, \cdots, x_n\}$ be given by a degree d circuit C and a degree d ABP P respectively, where $d = O(n^{O(1)})$. Then we can compute in polynomial time a circuit C' that computes $f \circ h$ where the size of C' is polynomially bounded in the sizes of C and P.*

Proof As in the proof of Theorem 5.4 we can assume that both f and h are homogeneous polynomials of degree d. Let f_g denote the polynomial computed at gate g of circuit C. Let w bound the number of nodes in any layer of P. Let $\langle i, a\rangle$ denote the a^{th} node in the i^{th} layer of P for $0 \leq i \leq d, 1 \leq a \leq w$. Let $h_{(i,a),(j,b)}$ denote the polynomial computed by ABP P', where P' is same as P but with source node $\langle i, a\rangle$ and sink node $\langle j, b\rangle$. We now describe the circuit C' computing the polynomial $f \circ h$. In C' we have gates $\langle g, l, (i, a), (i + l, b)\rangle$ for $0 \leq l \leq d, 0 \leq i \leq d$, $1 \leq a, b \leq w$ associated with each gate g of C, such that at the gate $\langle g, l, (i, a), (i + l, b)\rangle$ the circuit C' computes

$$r^{\langle g,l\rangle}_{(i,a),(i+l,b)} = f_{\langle g,l\rangle} \circ h_{(i,a),(i+l,b)} \tag{5.1}$$

where $f_{\langle g,l\rangle}$ denotes the degree l homogeneous component of the polynomial f_g.

If g is a $+$ gate of C with input gates g_1, g_2 so that $f_g = f_{g_1} + f_{g_2}$, we have $r^{\langle g,l\rangle}_{(i,a),(i+l,b)} = r^{\langle g_1,l\rangle}_{(i,a),(i+l,b)} + r^{\langle g_2,l\rangle}_{(i,a),(i+l,b)}$, for $0 \leq l \leq d$, $0 \leq i \leq d, 1 \leq a, b \leq w$. In other words, $\langle g, l, (i,a), (i+l,b)\rangle$ is a $+$ gate in C' with input gates $\langle g_1, l, (i,a), (i+l,b)\rangle$ and $\langle g_2, l, (i,a), (i+l,b)\rangle$. If g is a $\times$ gate in C we will have

$$r^{\langle g,l\rangle}_{(i,a),(i+l,b)} = \sum_{j=0}^{l}\sum_{t=1}^{w} r^{\langle g_1,j\rangle}_{(i,a),(i+j,t)} \cdot r^{\langle g_2,l-j\rangle}_{(i+j,t),(i+l,b)} \tag{5.2}$$

The above formula is easily computable by a small subcircuit. The output gate of C' will be $\langle g, d, (0,1), (d,1)\rangle$, where g is the output gate of C, and $(0,1)$ and $(d,1)$ are the source and the sink of the ABP P respectively. This is the description of the circuit C'. We inductively argue that gate $\langle g, l, (i,a), (i+l,b)\rangle$ of C' computes the polynomial $f_{\langle g,l\rangle} \circ h_{(i,a),(i+l,b)}$. If g is a $+$ gate of C the claim is obvious. Suppose g is a $\times$ gate of C with inputs g_1, g_2 such that $f_g = f_{g_1} \cdot f_{g_2}$. Inductively assume that the claim holds for the gates g_1 and g_2. Then we have $f_{\langle g,l\rangle} = \sum_{i=0}^{l} f_{\langle g_1,i\rangle} \cdot f_{\langle g_2,l-i\rangle}$. Hence, it easily follows that

$$\begin{aligned}
f_{\langle g,l\rangle} \circ h_{(i,a),(i+l,b)} &= \sum_{j=0}^{l}(f_{\langle g_1,j\rangle} \cdot f_{\langle g_2,l-j\rangle} \circ h_{(i,a),(i+l,b)}) \\
&= \sum_{j=0}^{l}\sum_{t=1}^{w} f_{\langle g_1,j\rangle} \cdot f_{\langle g_2,l-j\rangle} \circ h_{(i,a),(i+j,t)} \cdot h_{(i+j,t),(i+l,b)} \\
&= \sum_{j=0}^{l}\sum_{t=1}^{w}(f_{\langle g_1,j\rangle} \circ h_{(i,a),(i+j,t)}) \cdot (f_{\langle g_2,l-j\rangle} \circ h_{(i+j,t),(i+l,b)})
\end{aligned}$$

By induction hypothesis we have $r^{\langle g_1,j\rangle}_{(i,a),(i+j,t)} = f_{\langle g_1,j\rangle} \circ h_{(i,a),(i+j,t)}$ and $r^{g_2,l-j}_{(i+j,t),(i+l,b)} = f_{\langle g_2,l-j\rangle} \circ h_{(i+j,t),(i+l,b)}$. Now, from Equation 5.2 it is easy to obtain the desired Equation 5.1. Therefore, at the output gate $\langle g, d, (0,1), (d,1)\rangle$ the circuit C' computes $f \circ h$. The size of C' is bounded by a polynomial in the sizes of C and P. ∎

On the other hand, suppose f and g individually have small circuit complexity. Does $f \circ g$ have small circuit complexity? Can we compute such a circuit for $f \circ g$ from circuits for f and g? We first consider these questions for monotone circuits. It is useful

to understand the connection between monotone noncommutative circuits and context-free grammars.

Theorem 5.10 *There are monotone circuits C and C' computing polynomials f and g in $\mathbb{Q}\{X\}$ respectively, such that the polynomial $f \circ g$ requires monotone circuits of size exponential in $|X|$,* $\mathrm{size(C)}$*, and* $\mathrm{size(C')}$.

Proof Let $X = \{x_1, \cdots, x_n\}$. Define the finite language $L_1 = \{zww^r \mid z, w \in X^*, |z| = |w| = n\}$ and the corresponding polynomial $f = \sum_{m_\alpha \in L_1} m_\alpha$. Similarly let $L_2 = \{ww^r z \mid z, w \in X^*, |z| = |w| = n\}$, and the corresponding polynomial $g = \sum_{m_\alpha \in L_2} m_\alpha$. It is easy to see that there are $\mathrm{Poly}(n)$ size *unambiguous* acyclic CFGs for L_1 and L_2. Hence, by Proposition 2.134.14 there are monotone circuits C_1 and C_2 of size $\mathrm{Poly}(n)$ such that C_1 computes polynomial f and C_2 computes polynomial g. We first show that the finite language $L_1 \cap L_2$ cannot be generated by any acyclic CFG of size $2^{o(n \lg n)}$. Assume to the contrary that there is an acyclic CFG $G = (V, T, P, S)$ for $L_1 \cap L_2$ of size $2^{o(n \lg n)}$. Notice that $L_1 \cap L_2 = \{t \mid t = ww^r w, w \in X^*, |w| = n\}$.

Consider any derivation tree T' for a word $ww^r w = w_1 w_2 \ldots w_n w_n w_{n-1} \ldots w_2 w_1 w_1 \ldots w_n$. Starting from the root of the binary tree T', we traverse down the tree always picking the child with larger yield. Clearly, there must be a nonterminal $A \in V$ in this path of the derivation tree such that $A \Rightarrow^* u$, $u \in X^*$ and $n \leq |u| < 2n$. Crucially, note that any word that A generates must have same length since every word generated by the grammar G is in $L_1 \cap L_2$ and hence of length $3n$. Let $ww^r w = s_1 u s_2$ where $|s_1| = k$. As $|u| < 2n$, the string $s_1 s_2$ completely determines the string $ww^r w$. Hence, the nonterminal A can derive at most one string u. Furthermore, this string u can occur in at most $2n$ positions in a string of length $3n$. Notice that for each position in which u can occur it completely determines a string of the form $ww^r w$. Therefore, A can participate in the derivation of at most $2n$ strings from $L_1 \cap L_2$. Since there are n^n distinct words in $L_1 \cap L_2$, it follows that there must be at least $\frac{n^n}{2n}$ *distinct* nonterminals in V. This contradicts the size assumption of G.

Since $L_1 \cap L_2$ cannot be generated by any acyclic CFG of size $2^{o(n \log n)}$, it follows that the polynomial $f \circ g$ can not be computed by any monotone circuit of $2^{o(n \log n)}$ size. ∎

Theorem 5.10 shows that the Hadamard product of monotone circuits is more expressive than monotone circuits. It raises the question whether the permanent polynomial can be expressed as the Hadamard product of polynomial-size (or even subexponential size) monotone circuits. We note here that the permanent *can* be easily expressed as the Hadamard product of $O(n^3)$ many monotone circuits (in fact, monotone ABPs).

Theorem 5.11 *Suppose there is a deterministic subexponential-time algorithm that takes two circuits as input, computing polynomials f and g in $\mathbb{Q}\{x_1, \cdots, x_n\}$, and outputs a circuit for $f \circ g$. Then either* NEXP *is not in* $P/$Poly *or the Permanent does not have polynomial size noncommutative circuits.*

Proof Let C_1 be a circuit computing some polynomial $h \in \mathbb{Q}\{x_1, \ldots, x_n\}$. By assumption, we can compute a circuit C_2 for $h \circ h$ in subexponential time. Therefore, h is identically zero iff $h \circ h$ is identically zero iff C_2 evaluates to 0 on the all 1's input. We can easily check if C_2 evaluates to 0 on all 1's input by substitution and evaluation. This gives a deterministic subexponential time algorithm for testing if h is identically zero. By the noncommutative analogue of [KI03], shown in [AMS08], it follows that either NEXP $\not\subset P/$Poly or the Permanent does not have polynomial size noncommutative circuits. ∎

Next, We show that the identity testing problem: given $f, g \in \mathbb{F}\{X\}$ by circuits test if $f \circ g$ is identically zero is coNP hard.

Theorem 5.12 *Given two* monotone polynomial-degree *circuits C and C' computing polynomial $f, g \in \mathbb{Q}\{X\}$ it is* coNP*-complete to check if $f \circ g$ is identically zero.*

Proof We first show that the complement of the problem is in NP. The NP machine will guess a monomial $m_\alpha \in X^*$, $X = \{x_1, \ldots, x_n\}$ and check if coefficient of m_α is nonzero in both C and C'. Note that we can compute the coefficient of m_α in C and C' in deterministic polynomial time [AMS08]. Denote by CFGINT the problem of testing non emptiness of the intersection of two acyclic CFGs that generate Poly(n) length strings. By Lemma 4.14 CFGINT is polynomial time many-one reducible to testing if $f \circ g$ is identically zero. The problem of testing if the intersection of two CFGs (with

recursion) is nonempty is known to be undecidable via a reduction from the Post Correspondence problem [HMU, Chapter 9,Page 422]. We can give an analogous reduction from *bounded* Post Correspondence to CFGINT. The NP-hardness of CFGINT follows from the NP-hardness of bounded Post Correspondence [GJ79]. ∎

5.5 Overview

In this chapter we introduced and studied the Hadamard product of the multivariate polynomials in the free noncommutative polynomial ring $\mathbb{F}\{x_1, x_2, \cdots, x_n\}$. We explored arithmetic circuit and branching program complexity of the Hadamard product of polynomials when they are individually given by arithmetic circuits and/or algebraic branching programs.

When both of the polynomials are given by algebraic branching programs we gave a logspace algorithm to generate an ABP which computes the Hadamard product of the two polynomials. Using this result we showed that the identity testing problems for noncommutative ABPs over rationals is complete for the logspace counting class $\mathrm{C_{=}L}$ (which is known to be contained in NC^2). We have slightly weaker results for the identity testing problem for noncommutative ABPs over finite fields. We gave $\mathrm{Mod}_p\mathrm{L/Poly}$ and NC^3 upper bound on the complexity of the problem, where p is the characteristic of the field. It easily follows that the concerned identity testing problem is hard for NL. So it is difficult to improve above bound unconditionally to $\mathrm{Mod}_p\mathrm{L}$ (as it would show NL $\subseteq$ $\mathrm{Mod}_p\mathrm{L}$ which is an open problem). An interesting question in this context is: can we show deterministic NC^2 upper bound for the identity testing problem over noncommutative ABPs over finite fields.

We also explored the expressive power of the Hadamard product of two polynomials when either or both of them given by arithmetic circuit. We showed that if either of the two polynomials is given by an ABP the we can efficiently (in logspace) compute an arithmetic circuit for the Hadamard product of the polynomials. But if both the polynomials are give by arithmetic circuits then it is not easy to come up with an efficient algorithm to compute an arithmetic circuit for the Hadamard product of the two polynomials (such an algorithm would imply a non-trivial circuit-size lower bound).

Other question we studied is the following identity testing problem: Given two polynomials $f, g \in \mathbb{F}\{X\}$ either by an ABP or by an arithmetic circuit check whether $f \circ g$ is identically zero. We showed that if both the polynomials are given by arithmetic circuits then the problem is coNP-hard even when the circuits are monotone. Whereas, if either of the polynomials is given by an ABP the problem has polynomial time algorithm.

Bibliography

[Ajt98] M. AJTAI, The shortest vector problem in ℓ_2 norm is NP-hard for randomized reductions. *In proceedings of the 30th Annual ACM Symposium on theory of computing*, pages 10-19, Dallas, Texas.

[AC75] A. V. AHO, M. J. CORASICK, Efficient String Matching: An Aid to Bibliographic Search. *Commun. ACM,* 18(6): 333-340, 1975

[ABO99] E. ALLENDER, R. BEALS, AND M. OGIHARA, The complexity of matrix rank and feasible systems of linear equations, *Computational Complexity* , 8(2):99-126, 1999.

[ABSS97] S. ARORA, L. BABAI, J. STERN, E.Z. SWEEDYK, The hardness of approximate optima in lattices, codes, and system of linear equations. *Journal of Computer and System Sciences*, 54(2):317-331. Preliminary version in FOCS'93.

[AO96] E. ALLENDER, M. OGIHARA, Relationships among PL, #L and the determinant. *RAIRO - Theoretical Informatics and Applications*, 30:1–21, 1996.

[AS10] V. ARVIND, SRIKANTH SRINIVASAN, On the hardness of the noncommutative determinant. *STOC 2010*, 677-686.

[AJ08a] V. ARVIND AND PUSHKAR S. JOGLEKAR, Algorithmic Problems for Metrics on Permutation Groups.*In Proceedings of International Conference on Current trends in Theory and Practice of Computer Science, SOFSEM 2008:136-147.*

[AJ08b] V. ARVIND AND PUSHKAR S. JOGLEKAR, Some Sieving Algorithms for Lattice Problems*In Proceedings of IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2008.*

[AJ09] V. ARVIND AND PUSHKAR S. JOGLEKAR, Arithmetic Circuits, Monomial Algebras and Finite Automata *In Proceedings of International Symposium on Mathematical Foundations of Computer Science, MFCS 2008: 78-89.*

[AJS09] V. ARVIND, PUSHKAR S. JOGLEKAR AND SRIKANTH SRINIVASAN, Arithmetic Circuits and the Hadamard Product of Polynomial *In Proceedings of IARCS*

Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009.

[AKS01] M. AJTAI, R. KUMAR, D. SIVAKUMAR, A sieve algorithm for the shortest lattice vector. *In Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 266-275, 2001.

[AKS02] M. AJTAI, R. KUMAR, D. SIVAKUMAR, Sampling short lattice vectors and the closest lattice vector problem. *In Proceedings of the 17th IEEE Annual Conference on Computational Complexity-CCC*, 53-57, 2002.

[AM08] V. ARVIND, P. MUKHOPADHYAY Derandomizing the Isolation Lemma and Lower Bounds for Circuit Size.*APPROX-RANDOM,*276-289, 2008.

[AMS08] V. ARVIND, P. MUKHOPADHYAY, S. SRINIVASAN New results on Noncommutative Polynomial Identity Testing*In Proc. of Annual IEEE Conference on Computational Complexity,*268-279,2008.

[ARZ99] E. ALLENDER, K. REINHARDT, S. ZHOU, Isolation, matching and counting uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59(2):164–181, 1999.

[Bab86] L. BABAI, On Lovasz' lattice reduction and the nearest lattice point problem *Combinatorica*, 6(1), 1-13.

[Bh97] R. BHATIA, Matrix Analysis, Springer-Verlag Publishing Company, 1997.

[Bl00] J. BLÖMER, Closest vectors, successive minima, and dual HKZ-bases of lattices. *In Proceedings of th 17th ICALP,* Lecture Notes in Computer Science 1853, 248-259, Springer, 2000.

[Bos81] VAN EMDE BOAS, Another NP-complete problem and the complexity of computing short vectors in a lattice. *Technical Report 81-04*, Mathematische Instituut, University of Amsterdam.

[BL83] L. BABAI, E.M. LUKS, Canonical labeling of graphs. *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 171–183, 1983.

[BW05] A. BOGDANOV, H. WEE More on Noncommutative Polynomial Identity Testing *In Proc. of 20th Annual Conference on Computational Complexity,* 92-99, 2005.

[BCD79] I. F. BLAKE, G. COHEN, M. DEZA Coding with permutations *Information and Control 43*, 1-19, 1979.

[BCW06] C. BUCHHEIM, P.J. CAMERON, T. WU, On the Subgroup Distance Problem *ECCC*, TR06-146, 2006.

[BN07] J. BLÖMER, S. NAEWE Sampling Methods for Shortest Vectors, Closest Vectors and Successive Minima of lattices. *In Proceedings of ICALP,* 65-77, 2007.

[Ca88] P. J. CAMERON Metric and Geometric properties of sets of permutations *In Algebraic, Extremal and Metric Combinatorics*,39-53, 1988.

[CS04] S. CHIEN, A. SINCLAIR Algebras with polynomial identities and computing the determinant *In Proc. Annual IEEE Sym. on Foundations of Computer Science*,352-361, 2004.

[CW06] P.J. CAMERON, T. WU, The complexity of the Weight Problem for permutation groups. *Electronic Notes in Discrete Mathematics*, 2006.

[Cr85] D. E. CRITCHLOW, Metric methods for analyzing partially ranked data, *Lecture Notes in Statistics, Vol 34, Springer Verlag*, 1985

[DF77] M. DEZA, P. FRANKL, On the maximum number of permutations with given maximal or minimal distance *Journal of Combinatorial Theory (A)22*, 353-360, 1977.

[DH98] M. DEZA, T. HUANG, Metrics on Permutations, a Survey.*J. Combin. Inform. System Sci.* 23,173-185,1998.

[DFK91] M. DYER, A. FRIEZE, R. KANNAN A random polynomial time algorithm for approximating the volume of convex bodies. *Journal of the ACM* , 38(1):1-17, 1991.

[DMS99] I. DUMER, D. MICCIANCIO, M. SUDAN, Hardness of approximating minimum distance of a linear code. *40th Annual Symposium on Foundations of Computer Science*, 475-484, 1999.

[DPV10] D. DADUSH, C. PEIKERT, S. VEMPALA, Enumerative Algorithms for the Shortest and Closest Lattice Vector Problems in Any Norm via M-Ellipsoid Coverings. *Computer Science Arxive, arXiv:1011.5666v3 [cs.DS]*, 2010.

[DKS98] I. DINUR, G. KINDLER, S. SAFRA, Approximating CVP within almost polynomial factors is NP-hard. *In 39th Annual Symposium on Foundations of Computer Science*, Palo Alto, California. IEEE.

[GG00] O. GOLDREICH, S. GOLDWASSER On the limits of non approximability of lattice problems.*Comput. Syst. Sci*, 60(3):540-563,2000.

[GJ79] M. R. GAREY, D. S. JOHNSON Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman.*p. 228. ISBN 0-7167-1045-5*, 1979.

[GMSS99] O. GOLDREICH, D. MICCIANCIO, S. SAFRA, J. P. SEIFERT, Approximating shortest lattice vector is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71(2):55-61, 1999.

[HMU] J. E. HOPCROFT, R. MOTAWANI, J. D. ULLMAN, Introduction to Automata Theory Languages and Computation,*Second Edition*, Pearson Education Publishing Company.

[HR07] I. HAVIV, O. REGEV, Tensor-based Hardness of the Shortest Vector Problem to within Almost Polynomial Factors *STOC* 2007.

[IPZ01] R. IMPAGLIAZZO, R. PATURI, AND F. ZANE, Which Problems Have Strongly Exponential Complexity? *Journal Computer and System Sciences*, 63(4): 512-530 (2001).

[JS82] M. JERRUM, M. SNIR, Some Exact Complexity Results for Straight-Line Computations over Semirings.*J. ACM*, 29(3): 874-897, 1982.

[Kan87] R. KANNAN Minkowski's convex body theorem and integer programing. *Mathematics of Operational Rearch* ,12(3):415-440, 1987.

[KB97] R. KOCH, N. BLUM, Greibach Normal Form Transformation, Revisited, *STACS*, 1997, 47-54

[KI03] V. KABANETS, R. IMPAGLIAZZO, Derandomization of polynomial identity test means proving circuit lower bounds, *In Proc. of 35th ACM Sym. on Theory of Computing*,355-364,2003.

[KvM02] A. KLIVANS, D. VAN MELKEBEEK, Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

[KS01] ADAM KLIVANS, DANIEL A. SPIELMAN, Randomness efficient identity testing of multivariate polynomials. *In Proceedings of STOC* 216-223, 2001.

[Lu93] E.M. LUKS, Permutation groups and polynomial time computations. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 11:139–175, 1993.

[LLL82] A. K. LENSTRA, H. W. LENSTRA, JR. AND L. LOVASZ, Factoring Polynomials with Rational Coefficients, *Mathematische Annalen*, 261:515-534, 1982.

[Mi08] D. MICCIANCIO, Efficient reductions among lattice problems,*SODA*,2008,84-93

[MG02] D. MICCIANCIO, S. GOLDWASSER, *Complexity of Lattice Problems. A Cryptographic Perspective*, Kluwer Academic Publishers, 2002.

[MV97] M. MAHAJAN, V. VINAY, A Combinatorial Algorithm for the Determinant, *SODA 1997*, 730-738.

[MV10] DANIELE MICCIANCIO, PANAGIOTIS VOULGARIS, A Deterministic Single Exponential Time Algorithm for Most Lattice Problems based on Voronoi Cell Computations , *ECCC - Electronic Colloquium on Computational Complexity, TR10-014.*

[MVV87] KETAN MULMULEY, UMESH V. VAZIRANI, VIJAY V. VAZIRANI, Matching Is as Easy as Matrix Inversion *In Proceedings of STOC*345-354, 1987.

[N91] N. NISAN Lower bounds for noncommutative computation *In Proc. of 23rd ACM Sym. on Theory of Computing*, 410-418, 1991.

[Re] O. REGEV, Lecture Notes - Lattices in Computer Science, http://www.cs.tau.ac.il/ odedr/teaching/lattices _fall_2004/index.html.

[R67] D. J. ROSENKRANTZ, Matrix equations and normal forms for context-free grammars, *JACM (14)*, 1967, 501-507.

[RS05] R. RAZ, A. SHPILKA Deterministic polynomial identity testing in non commutative models *Computational Complexity*,14(1):1-19, 2005.

[Sch94] C. P. SCHNORR, Block reduced lattice bases and successive minima. *Combinatorics, Probability and Computing*, 3: 507-522.

[Si45] C. L. SIEGEL Lectures on Geometry of Numbers. *Springer-Verlag publishing company*, 1988.

[T91] S. TODA, Counting Problems Computationally Equivalent to the Determinant, manuscript.

[V91] V. VINAY, Counting Auxiliary Pushdown Automata and Semi-unbounded Arithmetic Circuits, *Proc. 6th Structures in Complexity Theory Conference*,270-284, 1991.