Derandomizing PIT for ROABP and Isolation Lemma for Special Graphs

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

by

Rohit Gurjar

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

December, 2015

CERTIFICATE

It is certified that the work contained in the thesis entitled "Derandomizing PIT for ROABP and Isolation Lemma for Special Graphs", by "Rohit Gurjar", has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

Manindra Agrawal
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

December, 2015

Synopsis

Randomization is ubiquitous in computer science. Many computational problems have faster algorithms when usage of random bits is allowed. It is a fundamental question whether using randomness can drastically decrease the complexity of a problem or every problem which has a randomized algorithm also has a deterministic one with only a small overhead. It is widely believed that the later is true. But such a deterministic solution, or in other words, derandomization, is known only for some specific problems, for example, primality testing and undirected reachability. Another reason derandomization is one of the central questions in computational complexity theory, is that it has connections with circuit lower bounds. It is known that derandomization is equivalent to proving that some problems are hard to solve.

Polynomial Identity Testing (PIT) is a good candidate problem to study the derandomization question. The problem asks if a given multivariate polynomial is identically zero. The input polynomial is given implicitly by an arithmetic computational model which computes polynomials, for example, arithmetic circuits. PIT has a polynomial time randomized algorithm, whereas finding even a sub-exponential time deterministic algorithm remains an open question. Derandomizing PIT also has connections with arithmetic circuit lower bounds. The PIT problem is studied in two paradigms (i) whitebox: where one can use the input circuit and (ii) blackbox: where one cannot see the input circuit, but can only evaluate the polynomial at some points from the field.

In this thesis, we give new deterministic identity tests for a special input model called ROABP and its variants. A read-once oblivious arithmetic branching program (ROABP) is essentially an iterated matrix product, where each matrix has its entries as univariate polynomials in a distinct variable. We give the first quasi-polynomial time ($n^{O(\log n)}$), completely blackbox test for ROABP, using a new idea called *basis isolation*. Earlier results either had the assumption of small individual degree or used the order of the variables associated with the matrices. For the case of known variable order, we give improved results, which imply a polynomial time test for constant-width ROABP. Width of an ROABP is the dimension of the matrices involved in the matrix product. Commutative ROABP is another variant where we give better results. An ROABP is commutative if the underlying matrices are commutative.

Another interesting question in the context of derandomization is the perfect matching problem. The problem asks if, in a given graph, there exists a pairing of adjacent vertices such that each vertex is paired with exactly one vertex. It has randomized parallel algorithms, but no such deterministic algorithms are known. One of the randomized algorithms goes via a reduction to PIT and uses the celebrated Isolation Lemma of Mulmuley, Vazirani and Vazirani. The Isolation Lemma states that assigning random weights to the edges of a graph ensures that it has a unique minimum weight perfect matching, with a good probability. We derandomize this lemma for $K_{3,3}$-free or K_5-free bipartite graphs, i.e., we give a deterministic log-space construction of such a weight assignment for these graphs. Such a construction was previously known for planar bipartite graphs, of which $K_{3,3}$-free or K_5-free bipartite graphs are natural generalizations. Our result implies that the perfect matching problem for $K_{3,3}$-free or K_5-free bipartite graphs is in SPL. It also gives an alternate proof for an already known result – reachability for $K_{3,3}$-free or K_5-free graphs is in UL. Our results are actually for a general class of graphs defined via the *clique-sum* operation, which subsumes $K_{3,3}$-free or K_5-free graphs.

Acknowledgements

I am immensely thankful to my supervisors Manindra Agrawal and Nitin Saxena for their continuous guidance and encouragement throughout my PhD. Whatever confidence I have in doing research, I owe it to them.

Being a student of Prof. Agrawal has been a privilege. He has always encouraged me to pursue my own ideas. The most important thing that I learnt from him is to not be scared of hard problems and that ideas are always simple. It is due to him that I got so many opportunities to work with different people and to attend workshops. I also want to thank him and his family for hosting us on so many occasions.

The last two years, when I was working with Nitin, have been the most productive period of my PhD. He has always been very generous with his time and we have had so many instructive discussions. I have learnt many wonderful ideas and techniques from him. His working style and approach to research has been an inspiration.

Most of the work during my PhD has been a joint work with Arpita. A lot of which would not have been possible without her. More importantly, I would not have learnt as much as I did. I want to thank her for being an incredible friend and for always being there.

Prof. Thomas Thierauf has been another important source of guidance for me. Working with him has always been a pleasure. I have worked with him on many different things, though they do not make a part of this thesis. Due to his keenness for working on new things, I have learnt a variety of topics. Among many other things which I learnt from him is how to write better articles. I also want to thank him for being a wonderful host on my trips to Ulm. I want to thank my other collaborators, Jochen Messner, Simon Straub, and Stephen Fenner. From Jochen, I learnt the importance of perseverance and rigour.

I am thankful to Rahul Arora, Ashu Gupta, and Raghunath Tewari. A part of this thesis is a joint work with them.

I thank Nisheeth Vishnoi for hosting me at MSR Bangalore. It was a very productive period for me. I am grateful to Chandan Saha for inviting me to IISc Bangalore. Although it was a short visit, it was very educative. I thank Vineet Nair for all the interesting discussions on theory and other things. I am thankful to Ramprasad who introduced us to many topics in theory and whose enthusiasm for research is inspiring.

I wish to express my gratitude to the CSE department at IITK, where I realized that learning can be fun. Thanks especially to Prof. Piyush Kurur who introduced me to a lot of interesting topics, which eventually encouraged me to pursue research in complexity theory. I want to thank all my colleagues at IITK who made my stay wonderful, especially to Puru, Balwinder, Umair, Diptarka, Sumanta, Amit, Rishabh, and Shubham. Special thanks to Sudhanshu for being a source of support since our B.Tech. days. Finally, I thank all my friends and family members for their constant love and support.

I thank TCS for supporting me financially through their research scholar program.

Contents

List of Publications **xiii**

List of Figures **xv**

1 Introduction **1**

1.1 Polynomial Identity Testing 2

1.1.1 Arithmetic Circuits 2

1.1.2 Arithmetic Branching Programs 5

1.1.3 Our results on PIT 8

1.1.4 Comparison with Boolean Pseudorandomness 9

1.2 Zero testing for an XOR of ROBPs 10

1.3 Perfect Matching 11

1.3.1 Our results on Matching 13

1.4 Organization of the Thesis 16

2 Polynomial Identity Testing and Various Arithmetic Models **17**

2.1 Definitions and Notations 17

2.2 Polynomial Identity Testing 19

2.3 Sparse Polynomials 20

2.4 Depth-3 Circuits 22

2.4.1 Diagonal Circuits 22

2.4.2 Set-multilinear Circuits 23

2.5 Arithmetic Branching Programs 24

2.5.1 Read-once Oblivious ABP 25
2.5.2 Sparse-factor ROABP 26
2.5.3 Commutative ROABP 27

3 Hitting-sets via Basis Isolation 29

3.1 Previous Works 29
3.1.1 Whitebox Test for ROABP [RS05] 29
3.1.2 Blackbox Test for Set-multilinear Circuits [ASS13] 31
3.2 Hitting-set for ROABP via Basis Isolation 33
3.2.1 Basis Isolation for ROABP 37
3.3 Basis Isolation in Diagonal Circuits 42
3.3.1 Basis Isolation in Homogeneous Diagonal Circuits 43
3.4 Discussion 47

4 Testing Zeroness of an XOR of Read-once Ordered Branching Programs 49

4.1 Read-once Ordered Branching Programs 50
4.1.1 Characterization of an ROBP 51
4.2 Equivalence of Two ROBPs 53
4.3 XOR of constantly many ROBPs 55
4.4 PIT for Sum of constantly many ROABPs 57
4.5 Discussion 57

5 Better Hitting-sets for Special Cases of ROABP 59

5.1 Hitting-set for Known-order ROABP 59
5.1.1 Pseudorandom Generators for ROBP 60
5.1.2 Bivariate ROABP 62
5.1.3 n-variate ROABP 65
5.2 Commutative ROABP 68
5.3 Discussion 75

6 Parallel Complexity of Bipartite Matching 77

6.1 Preliminaries . . . 77

6.1.1 Complexity Classes . . . 77

6.1.2 Graphs and Matchings . . . 78

6.1.3 Matching Polytope . . . 79

6.2 RNC algorithm for SEARCH-PM [MVV87] . . . 80

6.3 An Approach towards Matching in Bipartite Graphs . . . 82

6.4 Discussion . . . 85

7 Derandomizing Isolation Lemma for $K_{3,3}$-free or K_5-free Bipartite Graphs 87

7.1 Preliminaries . . . 88

7.1.1 Log-space Operations on Graphs . . . 89

7.1.2 Biconnected Graphs . . . 90

7.2 Isolation in Bipartite Planar Graphs [DKR10] . . . 91

7.3 Our Techniques . . . 93

7.4 Graph Decomposition . . . 95

7.4.1 Clique-sum . . . 95

7.4.2 Component Tree . . . 96

7.5 Nonzero Circulation . . . 97

7.5.1 Components of a Cycle . . . 98

7.5.2 Weighting Scheme . . . 100

7.5.3 Complexity of the Weight Assignment . . . 107

7.5.4 Construction of the Working Tree . . . 108

7.6 $K_{3,3}$-free and K_5-free Graphs . . . 110

7.6.1 Matching Preserving Operation . . . 110

7.6.2 Component Tree . . . 111

7.7 Discussion . . . 117

8 Conclusion 119

Bibliography **121**

Index **129**

List of Publications

[AGKS15] **Hitting-sets for ROABP and Sum of Set-Multilinear Circuits**
with Manindra Agrawal, Arpita Korwar, and Nitin Saxena
SIAM Journal of Computing (SICOMP) 2015

[AGGT14] **Derandomizing Isolation Lemma for K33-free and K5-free Bipartite Graphs**
with Rahul Arora, Ashu Gupta, and Raghunath Tewari
Electronic Colloquium on Computational Complexity (ECCC), 21:161, 2014

Chapter 3 is based on a part of [AGKS15] and Chapter 7 is based on [AGGT14]. The work in Chapter 4, 5, and 6 is yet unpublished.

List of Figures

1.1 A circuit 3

1.2 A graph with a perfect matching 11

2.1 An arithmetic branching program 24

2.2 An ROABP 26

2.3 A commutative ROABP 27

4.1 An ROBP 50

6.1 Minimum weight perfect matchings in a non-bipartite graph 83

7.1 The graphs $K_{3,3}$ and K_5 88

7.2 Clique-sum operation 95

7.3 A graph with its component tree 97

7.4 Breaking a cycle into its component cycles 99

7.5 Vertex-split operation 110

7.6 The four-rung Möbius ladder V_8. 111

7.7 Spliting a separating pair which is shared by many components 114

7.8 Spliting a vertex which is shared by many separating sets 115

Chapter 1

Introduction

Randomness has a wide spread application in computer science. Not only use of randomness can give faster algorithms, it is also essential for the security of cryptosystems, distributed computing etc. However, random numbers are hard to generate. Thus arises the question: how essential are random numbers? One of the central questions in computational complexity theory is that of derandomization. The question asks whether all problems with efficient randomized algorithms also have efficient deterministic algorithms, i.e., without using randomness. In terms of complexity classes, whether $\mathsf{P} = \mathsf{BPP}$. The answer is widely believed to be affirmative but little progress has been made towards proving that. This question is also intimately related to circuit lower bounds (see, for example, [AB09]).

The *polynomial identity testing (PIT)* problem acts as a good test case to study this question. The problem is to determine whether a given polynomial identity holds uniformly, or equivalently, whether a given polynomial is uniformly zero. For example, the following is a polynomial identity which holds uniformly.

$$(a^2 + b^2)(c^2 + d^2) = (ac - bd)^2 + (ad + bc)^2 \tag{1.1}$$

PIT has a polynomial time randomized algorithm [DL78, Zip79, Sch80] but there is no efficient deterministic algorithm known for it. The randomized algorithm is not only efficient but also very simple to describe. Given a polynomial $P(x_1, x_2, \ldots, x_n)$, evaluate it over a random point in $\mathbb{F}^n$. If the polynomial is zero then it always evaluates to zero but if it is nonzero then with a good probability, it evaluates to a nonzero value. Another motivation to study PIT comes from the fact that it has connections to arithmetic circuit lower bounds, i.e., the VP vs VNP question [KI03, Agr05]. PIT also has applications in other areas of complexity theory, for example, primality testing and perfect matching. In fact, the famous deterministic primality test by Agrawal, Kayal and Saxena [AKS02] involved derandomizing a special case of PIT. See the surveys by Saxena [Sax09, Sax14] and Shpilka & Yehudayoff [SY10] for more applications.

The perfect matching problem also reduces to PIT [Tut47], which gives a randomized NC (parallel) algorithm for perfect matching [Lov79]. Derandomizing this algorithm remains a challenging open question. In this thesis, we derandomize the PIT problem for some special input models and also derandomize the parallel algorithm for matching for some special graphs. The former involves algebraic techniques, while the latter involves graph theoretic techniques.

1.1 Polynomial Identity Testing

1.1.1 Arithmetic Circuits

The complexity of the PIT question depends crucially on the way the input polynomial is given. For example, if the polynomial is given as a set of monomials and corresponding coefficients, then we can easily check whether the polynomial is nonzero by checking the coefficient for each given monomial. The question becomes non-trivial when the polynomial is given implicitly, for example, by an arithmetic circuit. An arithmetic circuit is a natural computational model for polynomials. Arithmetic circuits are the arithmetic analogues of Boolean circuits and are defined over a field $\mathbb{F}$. They are directed acyclic graphs, where every internal node is a "+" or "$\times$" gate and the leaves are input gates which take a constant from the field $\mathbb{F}$ or a variable from the set $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$. Every edge has a label from the underlying field $\mathbb{F}$. The computation is done in the natural way: Each edge takes the polynomial computed at its tail and gives it to the head after

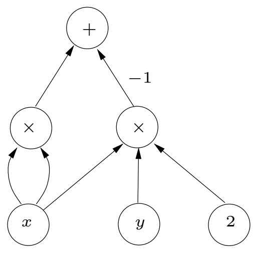

Figure 1.1: A circuit computing the polynomial $x^2 - 2xy$ (default edge label is 1).

multiplying with the edge label. Each + node computes the sum of all polynomials coming from all its incoming edges. Similarly, each $\times$ node computes the product of all polynomials coming from all its incoming edges. The polynomial computed at the output gate is said to be the polynomial computed by the circuit. Clearly, the output gate computes a polynomial in $\mathbb{F}[\boldsymbol{x}]$. We can restate the PIT problem as follows: Given an arithmetic circuit $\mathcal{C}$, decide whether the polynomial computed by $\mathcal{C}$ is zero, in time polynomial in the circuit size and the degree of the polynomial. Note that given a circuit, computing the polynomial explicitly is not possible in polynomial time, as it can have exponentially many monomials. However, given the circuit, it is easy to compute an evaluation of the polynomial by substituting the variables with constants.

The PIT problem is studied in two paradigms: whitebox and blackbox. Whitebox tests are ones which can use the structure of the given circuit. On the other hand a blackbox test cannot see the circuit, it can just evaluate the polynomial at some points from the field (or an extension field). In both the cases, circuit size is the input size. Observe that if the blackbox algorithm finds a point where the polynomial evaluates to a nonzero value, then it can stop and output that the polynomial is nonzero. So, in the blackbox paradigm, essentially one needs to produce a set of points such that if the polynomial is nonzero, it must evaluate to a nonzero value at one of the points in the set. This set is called a *hitting-set.* The terms blackbox test and hitting-set are used interchangeably in this thesis. Clearly, finding a hitting-set is harder than finding a whitebox test. Hitting-sets have stronger connections with arithmetic circuit lower bounds. In particular, Heintz and Schnorr [HS80] and later Agrawal [Agr05] showed that polynomial-time constructible hitting-sets for arithmetic circuits imply exponential size lower bounds for arithmetic circuits. Some results in other direction are also known, i.e., lower bounds imply derandomization of PIT [DSY09]. This connection between PIT and lower bounds suggests that derandomizing PIT is a hard problem, and indeed, it has been done only for very restricted classes of input models.

For the purpose of PIT one can assume, without loss of generality, that the given arithmetic circuit has alternating layers of $+$ and $\times$ gates. There is an easy reduction for this with only polynomial blow up in the circuit size. Moreover, one can assume that the output gate is a $+$ gate. This is because if it was a $\times$ gate, testing non-zeroness of the output polynomial reduces to testing non-zeroness of the factor polynomials. The first non-trivial deterministic test found was a polynomial time blackbox test for polynomials computed by depth-2 ($\Sigma\Pi$) circuits [BOT88, KS01]. Observe that a $\Sigma\Pi$ circuit is a sum of monomials, hence, there is an easy whitebox test: check the coefficient of each monomial. Next step would be solve PIT for depth-3 ($\Sigma\Pi\Sigma$) circuits. However, even a sub-exponential time whitebox test is not known for them till now. In a recent surprising result [GKKS13], it was shown that PIT for depth-3 circuits is almost as hard as that for general circuits. Earlier, such a result was known for depth-4 circuits [AV08, Koi12, Tav13]. Gupta, Kayal, Kamath and Saptharishi [GKKS13] showed that polynomial time blackbox PIT for depth-3 circuits would imply quasi-polynomial ($2^{(\log n)^{O(1)}}$) time blackbox PIT for general circuits with polynomially bounded degree. Hence, depth-3 circuits have become a stepping stone for understanding general circuits.

Recently, a sub-exponential time ($n^{\tilde{O}(n^{2/3})}$) hitting-set was given for depth-3 multilinear circuits by de Oliveira, Shpilka and Volk [dOSV15] [1]. A polynomial is said to be multilinear if the degree of every variable in every term is at most 1. A circuit is a multilinear circuit if the polynomial computed at every gate is multilinear. Since there are exponential lower bounds known for depth-3 multilinear circuits [RY09], a polynomial time PIT for them seems within reach.

There are some other special cases of depth-3 circuits, for which an efficient PIT is known. These are (i) depth-3 circuits with constant top fan-in, i.e., indegree of the output

[1]They also give hitting-sets for depth-4 multilinear circuits and regular formulas

(+) gate, (ii) diagonal depth-3 circuits, and (iii) depth-3 set-multilinear circuits.

After a long chain of work [DS07, KS07, KS09, KS11, SS11, SS12, SS13], a $\mathsf{poly}(n)d^k$-time blackbox test was given for depth-3 circuits with top fan-in k and degree d.

A diagonal depth-3 circuit, defined by Saxena [Sax08], is of the form $\sum_{i=1}^{k}(a_{i0} + \sum_{j=1}^{n} a_{ij}x_j)^{d_i}$. In other words, it is a depth-3 circuit with the multiplication gates being power gates. Saxena [Sax08] reduced diagonal circuits to a sum of products of univariate polynomials using the *duality trick*, which had a whitebox PIT due to Raz and Shpilka [RS05]. Later, blackbox tests were also obtained for diagonal circuits [FS13b, ASS13, FS13a, FSS14]. The best blackbox test for diagonal circuits has time complexity $n^{O(\log\log k)}$, which was given by Forbes, Saptharishi and Shpilka [FSS14].

Depth-3 set-multilinear circuits also have a polynomial time whitebox PIT due to Raz and Shpilka [RS05]. Consider a depth-3 multilinear circuit $\sum_{i=1}^{k}\prod_j \ell_{ij}$, where ℓ_{ij} is a linear polynomial for each i, j. Since every product gate here computes a multilinear polynomial, the linear polynomials $\{\ell_{ij}\}_j$ must be in disjoint sets of variables for each i. Thus, each product gate naturally induces a *partition* of the variables, where each *color* (i.e., part) of the partition contains the variables present in one of the linear polynomials. The circuit is called a depth-3 set-multilinear circuit, if the partitions induced by all the product gates are the same. Forbes and Shpilka [FS12] gave a $n^{O(\log k)}$-time blackbox PIT for set-multilinear circuits, when the partition induced by the product gates is known. Later, Agrawal, Saha and Saxena [ASS13] gave a complete blackbox PIT with the same time complexity.

All the three models described above are incomparable with each other. This thesis presents new results on PIT for read-once oblivious arithmetic branching programs (ROABP), a model which subsumes both depth-3 diagonal circuits and set-multilinear circuits. In the following section, we define ROABP and discuss previous results on it.

1.1.2 Arithmetic Branching Programs

An arithmetic branching program (ABP) is another interesting model for computing polynomials. It consists of a directed acyclic graph with a source and a sink. The edges of the graph have polynomials as their weights. Conventionally, the edge weights are taken to be 'simple' polynomials to restrict the power of the ABP. The weight of a path is defined to be the product of the weights of the edges present in the path. The polynomial computed by the ABP is the sum of the weights of all the paths from the source to the sink. It is well known that for an ABP, the underlying graph can be converted to a layered graph such that all paths from the source to the sink have exactly one edge in each layer. Then the polynomial computed by the ABP can be written as a *matrix product*, where each matrix corresponds to a layer. The entries in the matrices are weights of the corresponding edges. The maximum number of vertices in a layer, or, equivalently, the dimension of the corresponding matrices, is called the *width* of the ABP.

It is known that ABPs have the same expressive power as that of projections of symbolic determinant, i.e., determinant of a matrix with its entries being 'simple' polynomials [Ber84, Tod91, MV97]. Ben-Or and Cleve [BOC92] have shown that a polynomial computed by a formula can also be computed by a width-3 ABP of size polynomial in the formula size. A formula is a circuit with every node (except the input gates) having outdegree at most 1. Moreover, Saha, Saptharishi and Saxena [SSS09] reduce PIT for depth-3 circuits to PIT for width-2 ABP. Thus, ABP is a strong model for computing polynomials. The following chain of reductions shows the power of ABP and its constant-width version relative to other arithmetic computation models (see [BOC92] and [Nis91, Lemma 1]).

$$\begin{gathered}\text{Constant-depth arithmetic circuits} \leq_p \text{constant-width ABP}\\ =_p \text{formulas} \leq_p \text{ABP} \leq_p \text{arithmetic circuits}\end{gathered}$$

As mentioned earlier, some of the results in this thesis are for a special class of ABP called *read-once oblivious arithmetic branching programs (ROABP).* An ABP is a read-once oblivious ABP (ROABP) if the weights in its n layers are univariate polynomials in n distinct variables, i.e., the i-th layer has weights from $\mathbb{F}[x_{\pi(i)}]$, where π is a permutation on the set $\{1, 2, \dots, n\}$. In terms of matrix product, an ROABP can be written as $C(\boldsymbol{x}) = U^{\mathsf{T}}(\prod_{i=1}^{n} D_i)T$, where $U, T \in \mathbb{F}^{w \times 1}$ and $D_i \in \mathbb{F}^{w \times w}[x_{\pi(i)}]$ is a matrix with entries as polynomials in variable $x_{\pi(i)}$ for each i. When we know this permutation π, we call it an

ROABP with *known* variable order (it is significant only in the blackbox setting).

Raz and Shpilka [RS05] gave a $\mathsf{poly}(n, w, d)$-time whitebox algorithm for n-variate polynomials computed by a width-w ROABP with individual degree bound d. Forbes and Shpilka [FS13b] gave a $(nwd)^{O(\log n)}$-time blackbox algorithm for the same, when the variable order is known. Subsequently, Forbes, Saptharishi and Shpilka [FSS14] gave a blackbox test for the case of unknown variable order, but with time complexity $n^{O(d \log w \log n)}$. Note the exponential dependence on the individual degree. Their time complexity becomes quasi-polynomial in case of multilinear polynomials, i.e., $d = 1$ (in fact, even when $d = \mathsf{poly}(\log n)$).

Another model for which we present a hitting-set is Commutative ROABP, which is a special case of ROABP. A commutative ROABP is an ROABP where the corresponding matrix product is commutative. For example, when the matrices are diagonal matrices, which corresponds to the circuit model sum-of-products-of-univariate-polynomials. All the PIT results for ROABP (even with known order) also hold for commutative ROABP. Forbes, Sapthatrishi and Shpilka [FSS14] gave an improved hitting-set for commutative ROABP with time complexity $d^{O(\log w)}(nw)^{O(\log \log w)}$.

In another work, Jansen, Qiao and Sarma [JQS10b] gave a quasi-polynomial time blackbox test for a sum of constantly many multilinear "ROABP". Their definition of "ROABP" is more stringent. They assume that every variable appears at most once in the ABP. Later, this result was generalized to "read-r OABP" [JQS10a], where a variable can occur in at most one layer, and on at most r edges. Our definition of ROABP seems much more powerful than both of these.

ROABPs also subsume read-once formulas [Nis91]. A read-once formula is a formula where every variable occurs at most once. A non-trivial read-once formula always computes a nonzero polynomial, as there cannot be any cancellations. Thus, the whitebox test becomes trivial. However, a polynomial time hitting-set is not known for them. The best known time complexity is $n^{O(\log n)}$ [SV09].

1.1.3 Our results on PIT

This thesis gives new hitting-sets for the following classes of circuits/branching programs.

- $(ndw)^{O(\log n)}$-time hitting-set for ROABP.
- $\mathsf{poly}(n,d)$-time hitting-set for constant width ROABP, with known variable order (only for zero or large characteristic fields).
- $(ndw)^{O(\log\log w)}$-time hitting-set for commutative ROABP.

We elaborate these results further. The first result improves upon the ROABP result of [FSS14] and matches the time complexity for the unknown-order case with the known-order case (given by [FS13b]). Unlike the result of [FSS14], our result does not have an exponential dependence on the individual degree. Hence, our result can be seen as the first completely blackbox, quasi-polynomial time PIT for ROABP. Formally, we have the following theorem.

Theorem (Theorem 3.7)**.** *Let $C(\boldsymbol{x})$ be an n-variate polynomial computed by a width-w ROABP (unknown order) with the degree of each variable bounded by d. Then there is a $\mathsf{poly}(n,w,d)^{\log n}$-time hitting-set for C.*

Remark. Our algorithm also works when the layers have their weights as general sparse polynomials (still over disjoint sets of variables) instead of univariate polynomials (see the detailed version in Section 3.2).

Our result also implies an $(nk)^{O(\log n)}$-time hitting-set for set-multilinear circuits, but the hitting-set of Agrawal, Saha and Saxena [ASS13] is still better with the time complexity $n^{O(\log k)}$.

We also apply the same technique to diagonal circuits to get a hitting-set of size $(ndk)^{O(\log d)}$, where k is the top fan-in. This is incomparable with the $(ndk)^{\log\log k}$-size hitting-set of Forbes, Saptharishi and Shpilka [FSS14].

Our next result further improves the hitting-set size for known-order ROABP. However, it works only for zero or large characteristic fields.

Theorem (Theorem 5.6). *Let $C(\boldsymbol{x}) \in \mathbb{F}[\boldsymbol{x}]$ be an n-variate, individual degree d polynomial computed by a width-w ROABP in the variable order $(x_1, x_2, \ldots, x_n)$. Then there is a $dn^{O(\log w)}$-time hitting-set for C, when* $\operatorname{char}(\mathbb{F}) = 0$ *or* $\operatorname{char}(\mathbb{F}) > ndw^{\log n - 1}$.

Note that for constant width ROABPs, this gives the first polynomial time hitting-set (known-order). Our next result is for commutative ROABPs, which improves upon the previously best known hitting-set of size $d^{O(\log w)}(nw)^{O(\log \log w)}$ [FSS14].

Theorem (Theorem 5.15). *There is an $(ndw)^{O(\log \log w)}$-time hitting-set for n-variate commutative ROABPs with width w and individual degree d.*

1.1.4 Comparison with Boolean Pseudorandomness

Another motivation to study ROABPs comes from their Boolean analogues, called read-once ordered branching programs (ROBP). ROBPs have been studied extensively, with regard to the RL versus L question (randomized log-space versus log-space). The problem of finding hitting-sets for ROABP can be seen as an analogue of finding pseudorandom generators (PRG) for ROBP. A pseudorandom generator for a Boolean circuit C is an algorithm which can produce a probability distribution (with a small sample space) such that for the circuit C, the distribution is indistinguishable from the uniform random distribution. Constructing a pseudorandom generator (PRG) for ROBP with $O(\log n)$ seed length would imply $\mathsf{RL} = \mathsf{L}$. However, the best known pseudorandom generator (PRG) is of seed length $O(\log^2 n)$ ($n^{O(\log n)}$ size sample space), when variable order is known [Nis90, INW94, RR99]. This is the best result even for constant width ROBP. On the other hand, in the unknown-order case, the best known seed length is of size $n^{1/2+o(1)}$ [IMZ12].

There are a number of cases where pseudorandomness constructions for ROABP and ROBP, i.e., hitting-sets and PRGs, are quite similar in terms of complexity.

- ROABP with known variable order have $n^{O(\log n)}$-time hitting-set [FS13b], while known-order ROBP have $O(\log^2 n)$-seed PRG [Nis90, INW94].

- constant-width invertible ROABP have $\mathsf{poly}(n)$-time hitting-sets [AGKS15], while constant-width permutation ROBP have $O(\log n)$-seed PRG [KNP11, De11, Ste12].

- width-2 ROABP have $\mathsf{poly}(n)$-time hitting-sets [AGKS15], while width-2 ROBP have $O(\log n)$-seed PRG [BDVY13].

Here, we are not counting our result for known-order ROABP (Theorem 5.6), as it does not work over all fields. However, the hitting-set complexity it gives is better than the Boolean setting result. Even in the unknown-order case, our results for the arithmetic setting are better. One can ask whether we can get similar results in the Boolean setting. That is, for the known-order case, $O(\log n)$-seed PRG for constant width ROBP and for the unknown-order case, $O(\log^2 n)$ seed PRG for ROBP.

1.2 Zero testing for an XOR of ROBPs

After ROABPs, the next natural question would be PIT for a sum of two ROABPs. In [GKST15], we gave a polynomial time whitebox algorithm to test if a sum of two ROABPs (possibly in different variable orders) is zero. The idea was inspired by a similar known result in the Boolean setting, i.e., testing whether an XOR of two ROBPs is the zero function. It has a polynomial time algorithm due to Savický and Wegener [SW97].

In [GKST15], we also solve PIT for sum of constantly many ROABPs. Using the same approach, we can test the zeroness of an XOR of constantly many ROBPs. The PIT results on sum of ROABPs can be found in Korwar's thesis [Kor15]. In this thesis, we present the results in the Boolean setting.

Theorem (Theorem 4.8)**.** *Given c ROBPs, possibly in different variable orders, with n variables and width w, computing the functions $f_1, f_2, \ldots, f_c$, one can decide in time* $\mathsf{poly}(n^c, w^{2^c})$ *whether $f_1 \oplus f_2 \oplus \cdots \oplus f_c = 0$.*

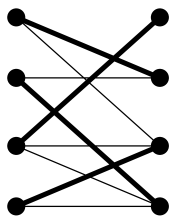

Figure 1.2: A graph with a perfect matching (represented by the bold edges)

1.3 Perfect Matching

The perfect matching problem is one of the most extensively studied problems in combinatorics, algorithms and complexity. In complexity theory, the problem plays a crucial role in the study of parallelization and derandomization. In a graph $G(V, E)$, a *matching* is a set of disjoint edges and a matching is called *perfect* if it covers all the vertices of the graph (Figure 1.2). Edmonds [Edm65] gave the first polynomial time algorithm for finding a perfect matching in a graph. Since then, there have been improvements in its sequential complexity [MV80]. However, there is no known NC (efficient parallel) algorithm for it. It is widely believed that the problem should be in NC as it has randomized NC (RNC) algorithms. A deterministic NC algorithm is known only for some special classes of graphs, for example planar graphs, regular bipartite graphs (see [KR98] for a detailed exposition on parallel complexity of matching).

There are various versions of the matching problem. Here, we are interested in the following two:

– Decision-PM: Decide if there exists a perfect matching in the given graph.

– Search-PM: Construct a perfect matching in the given graph, if it exists.

A randomized NC algorithm for Decision-PM was given by Lovász [Lov79]. Using the notion of Tutte Matrix [Tut47], Lovász [Lov79] reduced Decision-PM to PIT. For simplicity, we present the reduction only for bipartite graphs. A graph $G(V, E)$ is called bipartite if there exist two partitions of the vertex set V, say V_1 and V_2, such that any edge in the graph connects a vertex from V_1 to a vertex from V_2. For a bipartite graph to have a perfect matching, it must be that $|V_1| = |V_2|$. Let us say $V_1 = \{u_1, u_2, \ldots, u_n\}$ and

$V_2 = \{v_1, v_2, \ldots, v_n\}$. Consider the following $n \times n$ matrix B whose rows and columns are indexed by the vertices in the sets V_1 and V_2, respectively.

$$B(i,j) = \begin{cases} x_{ij} & \text{if } (u_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Here x_{ij}s are all distinct variables. Tutte [Tut47] showed that $\det(B)$, as a polynomial, is nonzero if and only if G has a perfect matching. The proof is simple. Any perfect matching M in G is permutation on the set $[n]$, i.e., there exists a permutation $\pi \colon [n] \to [n]$ such that $M = \{(u_i, v_{\pi(i)})\}_i$. As determinant is a signed sum over all permutations, i.e.,

$$\det(B) = \sum_{\pi \in S_n} \operatorname{sgn}(\pi) \prod_{i=1}^{n} B(i, \pi(i)),$$

$\det(B)$ has exactly as many distinct monomials as the number of perfect matchings in G.

To check if $\det(B)$ is nonzero, apply the randomized PIT, i.e., substitute random values for the variables $\{x_{ij}\}_{i,j}$. Note that determinant of matrix (over $\mathbb{F}$) can be computed efficiently. In fact, it can be computed in NC [Ber84, MV97]. Thus, we get an RNC algorithm for Decision-PM.

Subsequently, Search-PM was also shown to be in RNC [KUW86, MVV87]. The solution of Mulmuley, Vazirani and Vazirani [MVV87] was also based on PIT. Again for simplicity, we present their technique only for bipartite graphs. In the matrix B, they replace each variable x_{ij} with $2^{w_{ij}}$, where $\{w_{ij}\}_{i,j}$ are the weights assigned to the edges of the graph. It is easy to see that, after this replacement

$$\det(B) = \sum_{\substack{\pi \in S_n \\ \forall i \ (u_i, v_{\pi(i)}) \in E}} \operatorname{sgn}(\pi) \cdot 2^{\sum_i w_{i\pi(i)}}.$$

The quantity $\sum_i w_{i\pi(i)}$ is said to be the weight $w(M)$ of the matching M corresponding to the permutation π.

Now, the authors [MVV87] apply the powerful idea of the *Isolation Lemma*. The Isolation Lemma states that if the weights $\{w_{ij}\}_{i,j}$ are chosen randomly (from a polynomially bounded range), then with a good probability, a perfect matching is isolated, i.e., the minimum weight perfect matching is unique. This means that the term $2^{w(M)}$ corresponding to the minimum weight perfect matching will not cancel with any other term in $\det(B)$. Moreover, one can find the set of edges in the minimum weight perfect matching M as follows: delete an edge e and recompute $\det(B)$, if the term $2^{w(M)}$ does not survive, then $e \in M$. The procedure can be done for each edge in parallel.

Note that a perfect matching can be easily isolated by using exponentially large weights. But for an efficient computation of the determinant, it is necessary that the weights are polynomially bounded.

Derandomizing the isolation lemma, i.e., a deterministic NC construction of an isolating weight assignment, would put SEARCH-PM in NC. It remains a challenging open question. A general version of the Isolation Lemma has also been studied, where one has to ensure a unique minimum weight set in a (non-explicitly) given family of sets (or multi-sets). Here again, weight of a set S is $\sum_{e \in S} w(e)$. Derandomizing this general version would derandomize PIT. To elaborate, consider the set of monomials of a given polynomial as the given family of sets (viewing monomial as a multi-set of variables). If one can assign weights $\{w_i\}_i$ to the variables $\{x_i\}_i$ such that a monomial is isolated, then clearly, the replacement $x_i = t^{w_i}$ will keep the polynomial nonzero.

Using this connection with PIT, Arvind and Mukhopadhyay [AM08] have shown that derandomizing a version of the Isolation Lemma would imply circuit size lower bounds. While Reinhardt and Allender [RA00] have shown that derandomizing the Isolation Lemma for some specific families of paths in a graph would imply $\mathsf{NL} = \mathsf{UL}$.

1.3.1 Our results on Matching

We first present an approach for constructing an isolating weight assignment for bipartite graphs. For now, we do not have any weight construction which gives unique minimum weight perfect matching. So, the idea is to instead construct a weight assignment for a given graph G such that the number of minimum weight perfect matchings in G is significantly smaller than the total number of perfect matchings in G. Then work with the new graph G' generated from the union of all minimum weight perfect matchings. Our hope is that if the number of minimum weight perfect matchings is significantly reduced, then the new graph G' is also substantially smaller than G (in terms of number of edges). Observe that it is very much possible that in the graph G', other perfect matchings also appear which are not of the minimum weight. Indeed, there are examples of graphs, where G' remains same as G, even though not all perfect matchings in G are of minimum weight. It turns out that such examples are always non-bipartite graphs. In the bipartite case, we show that any perfect matching in G' is a minimum weight perfect matching of G. This implies that G' has to be smaller than G.

Theorem (Theorem 6.10)**.** *Let $G(V, E)$ be a bipartite graph with a weight function $w \colon E \to \mathbb{R}$ on its edges. Let E' be the union of all minimum weight perfect matchings in G according to w. Then every perfect matching in the graph $G' = (V, E')$ has the same weight – the minimum weight of any perfect matching in G.*

However, it remains an open question to construct a weight assignment which ensures that G' is significantly smaller than G, i.e., it has only a fraction of edges from G. If this can be done then one can repeat the same operation $O(\log n)$ times to reach a graph with just one perfect matching.

Next, we move on to special classes of graphs for which a perfect matching can be isolated. The Isolation Lemma has been derandomized for the following special classes of graphs: planar bipartite graphs [DKR10, TV12], constant genus[2] bipartite graphs [DKTV12], strongly chordal graphs [DK98], graphs with small number of perfect matchings [GK87, AHT07] and graphs with small number of nice cycles [Hoa10]. A graph is planar if it can be drawn on a plane without any edge crossings.

It is well known that a graph is planar if and only if it is both $K_{3,3}$-free and K_5-free [Wag37]. For a graph H, G is called an H-free graph if H is not a minor of G i.e., H cannot be formed from G by deleting edges and vertices and contracting edges. $K_{3,3}$ is the complete bipartite graph with $(3, 3)$ nodes and K_5 is the complete graph with 5 nodes. $K_{3,3}$ and K_5 are, in a sense, the smallest non-planar graphs. Wagner [Wag37] showed that any non-planar graph will have a $K_{3,3}$ or a K_5 as a minor. A natural generalization

[2]a planar graph has genus zero.

of planar graphs would be $K_{3,3}$-free graphs or K_5-free graphs. Note that a $K_{3,3}$-free graph can contain a K_5 as a minor or vice versa. We make a further step towards the derandomization of the Isolation Lemma by derandomizing it for two graph classes: $K_{3,3}$-free bipartite graphs and K_5-free bipartite graphs. Note that these graphs are not captured by the classes of graphs mentioned above. In particular, a $K_{3,3}$-free or K_5-free graph can have arbitrarily high genus, exponentially many perfect matchings, or exponentially many nice cycles.

Theorem (Theorem 7.1)**.** *Given a $K_{3,3}$-free or K_5-free bipartite graph, an isolating weight assignment (polynomially bounded) for it can be constructed in log-space.*

Another motivation to study these graphs came from the fact that COUNT-PM (counting the number of perfect matchings) is in NC^2 for $K_{3,3}$-free graphs [Vaz89] and in TC^2 ($\subseteq \mathsf{NC}^3$) for K_5-free graphs [STW14]. These were the best known results for DECISION-PM too. The counting results, together with the known NC-reduction from SEARCH-PM to COUNT-PM (for bipartite graphs) [KMV08], implied an NC algorithm for SEARCH-PM. Thus, a natural question was to find a direct algorithm for SEARCH-PM via isolation, which we do here. One limitation of the earlier approach is that COUNT-PM is $\#\mathsf{P}$-hard for general bipartite graphs. Thus, there is no hope of generalizing this approach to work for all graphs. While the isolation approach can potentially lead to a solution for general/bipartite graphs.

Theorem 7.1 puts DECISION-PM and SEARCH-PM for $K_{3,3}$-free or K_5-free bipartite graphs in the classes SPL and $\mathsf{FL}^{\mathsf{SPL}}$, respectively, which are subsets of NC^2 (see Chapter 7 for details).

The crucial property of these graphs, which we use, is that their 4-connected components are either planar or constant sized. This property has been used to reduce various other problems on $K_{3,3}$-free or K_5-free graphs to their planar version, e.g. graph isomorphism [DNTW09], reachability [TW14]. However, their techniques do not work directly for the matching problem. There has been an extensive study on more general minor-free graphs by Robertson and Seymour. In a long series of works, they gave similar decomposition properties for these graphs [RS03]. Our approach for matching can possibly be generalized to H-free graphs for a larger/general graph H.

1.4 Organization of the Thesis

In Chapter 2, we give an introduction to PIT and various arithmetic models. In Chapter 3, we introduce the technique of basis isolation and apply it to get new hitting-sets for ROABPs and diagonal circuits. In chapter 4, we give an algorithm to decide if an XOR of constantly many ROBPs computes the zero function. In Chapter 5, we give improved hitting-sets for two special cases of ROABPs, namely ROABPs with known variable order and commutative ROABP.

Next two chapters deal with the matching problem. In Chapter 6, we describe a well-known randomized parallel algorithm for matching and present some preliminary ideas towards derandomizing it. In Chapter 7, we do this derandomization for $K_{3,3}$-free or K_5-free bipartite graphs.

Chapter 2

Polynomial Identity Testing and Various Arithmetic Models

In this chapter, we give the basic definitions and notations used in this thesis. We give an introduction to polynomial identity testing and describe various arithmetic models and relations between them.

2.1 Definitions and Notations

In the context of time complexity, quasi-polynomial time means $2^{O((\log n)^c)}$-time for some fixed c, where n is the input size.

Throughout this thesis, $\mathbb{N}$ denotes the set of all non-negative integers, i.e., $\{0, 1, 2, \dots\}$. $[n]$ denotes the set $\{1, 2, \dots, n\}$. $[\![d]\!]$ denotes the set $\{0, 1, \dots, d\}$. $\boldsymbol{x}$ will denote a set of variables, usually the set $\{x_1, x_2, \dots, x_n\}$. For a set of n variables $\boldsymbol{x}$ and for an exponent $\boldsymbol{a} = (a_1, a_2, \dots, a_n) \in \mathbb{N}^n$, $\boldsymbol{x}^{\boldsymbol{a}}$ will denote the monomial $\prod_{i=1}^{n} x_i^{a_i}$. The *support* of a monomial $\boldsymbol{x}^{\boldsymbol{a}}$, denoted by $\mathrm{Supp}(\boldsymbol{a})$, is the set of variables appearing in that monomial, i.e., $\{x_i \mid i \in [n], a_i > 0\}$. The *support size* of a monomial is the cardinality of its support, denoted by $\mathrm{supp}(\boldsymbol{a})$. For a polynomial $P(\boldsymbol{x})$, the coefficient of a monomial $\boldsymbol{x}^{\boldsymbol{a}}$ in $P(\boldsymbol{x})$ is denoted by $\mathrm{coef}_P(\boldsymbol{x}^{\boldsymbol{a}})$. In particular, $\mathrm{coef}_P(1)$ denotes the constant term of the polynomial P.

For a monomial $\boldsymbol{x}^{\boldsymbol{a}}$, $\sum_i a_i$ is said to be its *degree* and a_i is said to be its *degree in variable* x_i for each i. Similarly for a polynomial P, its degree (or degree in x_i) is the maximum degree (or maximum degree in x_i) of any monomial in P with a nonzero coefficient. Formally,

$$\deg(P) = \max\left\{\sum_{i=1}^{n} a_i \mid \operatorname{coef}_P(\boldsymbol{x}^{\boldsymbol{a}}) \neq 0,\ \boldsymbol{a} \in \mathbb{N}^n\right\}.$$

$$\deg_{x_i}(P) = \max\left\{a_i \mid \operatorname{coef}_P(\boldsymbol{x}^{\boldsymbol{a}}) \neq 0,\ \boldsymbol{a} \in \mathbb{N}^n\right\}.$$

We define the *individual degree* of P to be $\text{indv-deg}(P) = \max\{\deg_{x_i}(P) \mid i \in [n]\}$.

To better understand some circuit models, we often use polynomials over an algebra $\mathbb{A}$, i.e., polynomials whose coefficients come from $\mathbb{A}$. An algebra is a vector space V, equipped with a vector product, i.e., a binary operation from $V \times V$ to V. The product is associative and distributive with the $+$ operation of the vector space. For two elements v_1, v_2 in algebra $\mathbb{A}$, $v_1 v_2$ denotes this vector product. The dimension of an algebra is the dimension of the underlying vector space. When this vector product is simply a coordinate-wise product, then the resulting algebra is called the Hadamard algebra. $\mathbb{H}_k$ denotes the k-dimensional Hadamard algebra.

Apart from this we will also consider the matrix algebra. Matrix algebra is just the vector space of matrices equipped with the matrix product. $\mathbb{F}^{m\times n}$ represents the set of all $m \times n$ matrices over the field $\mathbb{F}$. Note that the algebra of $w \times w$ matrices, has dimension w^2. Let $\mathbb{A}_k$ be any k-dimensional algebra over the field $\mathbb{F}$. For any two elements $A = (a_1, a_2, \ldots, a_k) \in \mathbb{A}_k$ and $B = (b_1, b_2, \ldots, b_k) \in \mathbb{A}_k$ (having a natural basis representation in mind), their dot product is defined as $A \cdot B = \sum_{i=1}^{k} a_i b_i$. To be clear, the definition also holds for the matrix algebra.

We often view a vector/matrix with polynomial entries, as a polynomial with vector/matrix coefficients. For example,

$$D(x,y) = \begin{pmatrix} 1+x & y-xy \\ x+y & 1+xy \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} 1 + \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} x + \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} y + \begin{pmatrix} 0 & -1 \\ 0 & 1 \end{pmatrix} xy.$$

The coef_D operator will give a matrix for any monomial, for example, $\mathrm{coef}_D(y) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. For a polynomial $D(\boldsymbol{x}) \in \mathbb{A}[\boldsymbol{x}]$ over an algebra, its *coefficient space* is the space spanned by its coefficients.

For a matrix R, $R(i,j)$ denotes its entry in the i-th row and j-th column. $R(i,\cdot)$ and $R(\cdot,i)$ denote the i-th row and the i-th column of R, respectively.

2.2 Polynomial Identity Testing

It is a well-known fact that a degree-d univariate polynomial over a field, can have at most d roots. Thus, there is a simple test for the non-zeroness of the polynomial: evaluate the polynomial at $d+1$ distinct points. A generalization of this fact to multivariate polynomials gives the following lemma, which forms the basis of an efficient randomized PIT.

Lemma 2.1 ([DL78, Sch80]). *Let $f(\boldsymbol{x}) \in \mathbb{F}[\boldsymbol{x}]$ be a degree d, n-variate polynomial over a field $\mathbb{F}$. Let $S \subseteq \mathbb{F}$ be a set of size $> d$. Then*

$$\Pr[f(r_1, r_2, \ldots, r_n) \neq 0] \geq 1 - \frac{d}{|S|},$$

where r_i is chosen uniformly randomly from S for each i, independently.

This version of the lemma is by Schwartz [Sch80], while DeMillo and Lipton gave a slightly weaker probability bound of $1 - nd/|S|$. Choosing the set S to be of size $2d$, one gets the success probability of $1/2$. Note that for this randomized test to work, we need the field size to be large enough. In case of finite fields, one can go to an appropriately large extension. For a blackbox PIT, it is essential that we allow evaluation points to be from a field extension. For example, let us consider the polynomial $f(x) = x^2 - x$ over $\mathbb{F}_2$ (the field of size 2). Clearly, $f(0) = f(1) = 0$. To get a point where $f(x)$ evaluates to a nonzero value, one must go to an extension of $\mathbb{F}_2$. Usually, a polynomial size extension suffices for blackbox PIT.

There is another version of Lemma 2.1, given by Zippel [Zip79]. It gives a better probability bound, when an upper bound is known on the individual degree of the polynomial, instead of the degree.

Lemma 2.2 ([Zip79]). *Let $f(\boldsymbol{x}) \in \mathbb{F}[\boldsymbol{x}]$ be an n-variate polynomial over a field $\mathbb{F}$ with* indv-deg$(f) = d$. *Let $S \subseteq \mathbb{F}$ be a set of size $> d$. Then*

$$\Pr[f(r_1, r_2, \ldots, r_n) \neq 0] \geq \left(1 - \frac{d}{|S|}\right)^n,$$

where r_i is chosen uniformly randomly from S for each i, independently.

Note that the randomized PIT works for any polynomial, with even exponentially high degree. In this case, the set S will have to be of exponential size, but a random element from S would need just $O(n)$ random bits. For a deterministic PIT, we always assume the degree to be small. In other words, a PIT is said to be efficient if its time complexity is polynomial in the circuit size and the degree.

As mentioned earlier, a deterministic blackbox PIT is equivalent to constructing a hitting-set.

Hitting-set: A set of points $\mathcal{H} \in \mathbb{F}^n$ is called a hitting-set for a class C of n-variate polynomials if for any nonzero polynomial P in C, there exists a point in $\mathcal{H}$ where P evaluates to a nonzero value.

An $f(n)$-time hitting-set would mean that the hitting-set can be generated in time $f(n)$ for input size n. From Lemma 2.2, it follows that any set S^n, with $|S| > d$, is a hitting-set for n-variate polynomials with individual degree d.

2.3 Sparse Polynomials

The polynomials computed by depth-2 ($\Sigma\Pi$) circuits are also called sparse polynomials, as their number of monomials is bounded by the circuit size. A polynomial is called s-*sparse* if there are at most s monomials in it with nonzero coefficients. As mentioned earlier, there are polynomial time blackbox tests for sparse polynomials (see, for example, [BOT88, KS01]). A usual technique for PIT is to come with a univariate monomial map which preserves non-zeroness, i.e., constructing a weight function $\mathrm{w}\colon \boldsymbol{x} \to \mathbb{N}$ such that when x_i is replaced with $t^{\mathrm{w}(x_i)}$ for each $i \in [n]$, the polynomial remains nonzero. Afterwards a hitting-set for univariate polynomials can be applied, i.e., substituting (degree +1) many values for t. We present one such construction for sparse polynomials.

The function w can be naturally extended to the set of all monomials as follows: $\mathrm{w}(\prod_{i=1}^{n} x_i^{\gamma_i}) = \sum_{i=1}^{n} \gamma_i \mathrm{w}(x_i)$, for any $(\gamma_i)_i \in \mathbb{N}^n$. Note that if each variable x_i is replaced with $t^{\mathrm{w}(x_i)}$ then any monomial m just becomes $t^{\mathrm{w}(m)}$. We say w separates a pair of monomials (m, m'), if $\mathrm{w}(m) \neq \mathrm{w}(m')$. There is a folklore trick which separates any given set of a small number of monomials. Clearly, separating all monomials of a sparse polynomial will give us a blackbox PIT. We present this trick in the following lemma. Later, it will also be used for designing hitting-sets for ROABPs (end of proof of Theorem 3.7).

Lemma 2.3 (Efficient Kronecker map [Kro82, AB03])**.** *Let M be the set of all monomials in n variables $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$ with maximum individual degree d. For any value s, there is a polynomial-time constructible set of $N := ns\log(d+1)$ weight functions from $\boldsymbol{x}$ to $[2N\log N]$, such that for any set $A \subseteq M^2$ of s pairs of monomials, at least one of the weight functions* w *separates all the pairs in A; i.e., for all $(m, m') \in A$, $\mathrm{w}(m) \neq \mathrm{w}(m')$.*

Proof. Since we want to separate the n-variate monomials with maximum individual degree d, we use the naïve Kronecker map $W\colon x_i \mapsto (d+1)^{i-1}$, for all $i \in [n]$. It can be easily seen that W will give distinct weights to any two monomials (with maximum individual degree d). But, the weights given by W are exponentially high.

So, we take the weight function W modulo p for many small primes p. Each prime p leads to a different weight function. That is our set of candidate weight functions. We need to bound the number N of primes which ensures that at least one of the weight functions separates all the monomial pairs in A. We choose the smallest N primes, say $\mathcal{P}$ is the set. By the effective version of the Prime Number Theorem, the highest value in the set $\mathcal{P}$ is less than $2N\log N$.

To bound the number N of primes: We want a $p \in \mathcal{P}$ such that for all $(m, m') \in$

$A,\ W(m) - W(m') \not\equiv 0 \pmod{p}$. Which means,

$$\exists p \in \mathcal{P},\ p \nmid \prod_{(m,m') \in A} \big(W(m) - W(m')\big)\,.$$

In other words,

$$\prod_{p \in \mathcal{P}} p \nmid \prod_{(m,m') \in A} \big(W(m) - W(m')\big)\,.$$

This can be ensured by setting $\prod_{p \in \mathcal{P}} p > \prod_{(m,m') \in A} (W(m) - W(m'))$. There are s such monomial pairs and each $W(m) < (d+1)^n$. Also, $\prod_{p \in \mathcal{P}} p > 2^N$. Hence, $N = ns \log(d+1)$ suffices. □

2.4 Depth-3 Circuits

Now, we define two special classes of depth-3 circuits, which will be discussed in this thesis and for which an efficient blackbox PIT is known.

2.4.1 Diagonal Circuits

A diagonal circuit is of the form

$$\sum_{i=1}^{k} (a_{i0} + a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n)^{d_i}\,,$$

where $a_{ij} \in \mathbb{F}$ for each i, j. As discussed before, Saxena [Sax08] showed that any polynomial computed by a diagonal circuit can also be computed by a sum of products of univariate polynomials, i.e.,

$$\sum_{i=1}^{t} f_{i1}(x_1) f_{i2}(x_2) \cdots f_{in}(x_n).$$

with only a polynomial blow-up in the size. The proof of Saxena works only for zero or large enough characteristic fields. Here, we present a proof by Shpilka (given in [Gup15]), which works over all characteristic fields.

Lemma 2.4 (Duality trick [Sax08]). *A polynomial* $f = \left(a_0 + \sum_{j=1}^{n} a_j x_j\right)^d$ *can be written* *as*

$$\sum_{i=1}^{t} f_{i1}(x_1) f_{i2}(x_2) \cdots f_{in}(x_n),$$

where $t = O(nd^2)$ and f_{ij} is a univariate polynomial for each i, j.

Proof. Consider the polynomial $P = (y + a_0)(y + a_1 x_1) \cdots (y + a_n x_n) - y^{n+1}$. Viewing P as a polynomial in y, its degree is n and $\mathrm{coef}_P(y^n) = a_0 + \sum_{j=1}^{n} a_j x_j$. It is easy to see that in the polynomial P^d, the coefficient of y^{dn} is $(a_0 + \sum_{j=1}^{n} a_j x_j)^d$.

To extract this coefficient one can use polynomial interpolation. It is well known that for a univariate polynomial, any of its coefficients can be written as a linear combination of its evaluations. In particular, for the polynomial P^d (with degree nd), there exist constants $\{\alpha_i\}_{i=1}^{nd+1}$ and $\{\beta_i\}_{i=1}^{nd+1}$ such that

$$\mathrm{coef}_{P^d}(y^{nd}) = \sum_{i=1}^{nd+1} \alpha_i P^d(\beta_i).$$

For any $i \in [nd+1]$, $P^d(\beta_i) = \left((\beta_i + a_0)(\beta_i + a_1 x_1) \cdots (\beta_i + a_n x_n) - \beta_i^{n+1}\right)^d$ can be expanded as

$$\sum_{r=0}^{d} \binom{d}{r} (-1)^{d-r} \beta_i^{(n+1)(d-r)} \cdot (\beta_i + a_0)^r (\beta_i + a_1 x_1)^r \cdots (\beta_i + a_n x_n)^r.$$

After this expansion, we get the desired form with $t = (d+1)(nd+1)$. $\square$

Applying the same trick to a sum of powers of linear polynomials, i.e., diagonal circuits, we get the desired reduction. Later, we will argue that a sum of products of univariate polynomials can also be computed by an ROABP, in fact by a commutative ROABP.

2.4.2 Set-multilinear Circuits

A set-multilinear circuit is of the form

$$C(\boldsymbol{x}) = \sum_{i=1}^{k} \prod_{j=1}^{q} \ell_{ij}(\boldsymbol{x}_j),$$

where $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_q$ are disjoint sets of variables and $\ell_{ij}(\boldsymbol{x}_j)$ is a linear polynomial in the variables $\boldsymbol{x}_j$ for each i, j. Let $\boldsymbol{x}_j = \{x_{j1}, x_{j2}, \ldots, x_{jn}\}$ and $\ell_{ij}(\boldsymbol{x}_j) = a_{ij0} + a_{ij1} x_{j1} + \cdots +$

$a_{ijn}x_{jn}$. Define vectors $v_{jn} \in \mathbb{F}^k$ as $v_{jn} = (a_{1jn}, a_{2jn}, \ldots a_{kjn})$. Then one can view the polynomial $f(\boldsymbol{x})$ as a dot product $(1, 1, \ldots, 1) \cdot D(\boldsymbol{x})$, where

$$D(\boldsymbol{x}) = \prod_{j=1}^{q} (v_{j0} + v_{j1}x_{j1} + \cdots + v_{jn}x_{jn}).$$

Here $D(\boldsymbol{x})$ is a polynomial over the k-dimensional Hadamard algebra $\mathbb{H}_k$. The blackbox test of Agrawal, Saha and Saxena [ASS13] use this view of set-multilinear circuits to give a hitting-set. As we will see later, for PIT, it is enough to find the space spanned by the coefficients of $D(\boldsymbol{x})$.

We can say that the polynomial $D(\boldsymbol{x})$ is computed by a $\Pi\Sigma$ circuit over the Hadamard algebra $\mathbb{H}_k$, i.e., the edge labels in the circuit are from $\mathbb{H}_k$. Moreover, a variable is input to only one of the $+$ gates. A natural generalization would be to consider a similar $\Pi\Sigma$ circuit over the matrix algebra. The resulting model will be ROABP, as we will see in the next section. In fact, we give a hitting-set for something called, a sparse-factor ROABP, which corresponds to a $\Pi\Sigma\Pi$ circuit over the matrix algebra.

2.5 Arithmetic Branching Programs

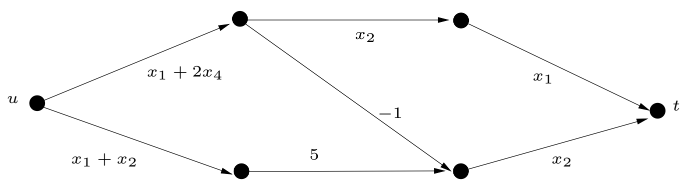

Figure 2.1: An ABP computing the polynomial $(x_1+2x_4)x_2x_1-(x_1+2x_4)x_2+5x_2(x_1+x_2)$.

An arithmetic branching program (ABP) is a directed acyclic graph, with a source vertex u and a sink vertex t, and the edges have polynomials as their weights (Figure 2.1). The polynomials on the edges are 'simple', for example, linear or univariate. For an edge e, let us denote its weight by $W(e)$. For a path p from u to t, its weight $W(p)$ is defined to be the product of weights of all the edges in it, i.e., $\prod_{e \in p} W(e)$. Consider the polynomial $C(\boldsymbol{x}) = \sum_{p \in \text{paths}(u,t)} W(p)$ which is the sum of the weights of all the paths from u to t. This polynomial $C(\boldsymbol{x})$ is said to be computed by the ABP.

One can make this directed acyclic graph layered by introducing new vertices and edges with weight 1. Thus, one can redefine an ABP as a directed graph with $q+1$ layers of vertices $\{V_0, V_1, \ldots, V_q\}$ and a start node u and an end node t such that the edges are only going from u to V_0, V_{i-1} to V_i for any $i \in [q]$, V_q to t. Edges have weights from $\mathbb{F}[\boldsymbol{x}]$ and as a convention, the edges going from u and coming to t have weights from the field $\mathbb{F}$. The polynomial computed by the ABP is defined in the same way. The ABP is said to have width w if $|V_i| \leq w$ for all $i \in [\![d]\!]$. Without loss of generality we can assume $|V_i| = w$ for each $i \in [\![d]\!]$.

It is well-known that sum over all paths in a layered graph can be represented by an iterated matrix multiplication. To see this, let the set of nodes in V_i be $\{v_{i,j} \mid j \in [w]\}$. It is easy to see that the polynomial computed by the ABP is the same as $U^{\mathsf{T}}(\prod_{i=1}^{q} D_i)T$, where $U, T \in \mathbb{F}^{w \times 1}$ and D_i is a $w \times w$ matrix for $1 \leq i \leq q$ such that

$$\begin{aligned} U(\ell) &= W(u, v_{0,\ell}) \text{ for } 1 \leq \ell \leq w \\ D_i(k,\ell) &= W(v_{i-1,k}, v_{i,\ell}) \text{ for } 1 \leq \ell, k \leq w \text{ and } 1 \leq i \leq q \\ T(k) &= W(v_{d,k}, t) \text{ for } 1 \leq k \leq w \end{aligned}$$

2.5.1 Read-once Oblivious ABP

An ABP is called a *read-once oblivious ABP (ROABP)* if the edge weights in different layers are univariate polynomials in distinct variables. Formally, the entries in D_i come from $\mathbb{F}[x_{\pi(i)}]$ for all $i \in [q]$, where π is a permutation on the set $[q]$. Here q is same as n, the number of variables.

The order $(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$ is said to be the variable order of the ROABP. The variable order of an ROABP is a crucial information. It is possible that a polynomial can be computed by small ROABP in some variable order, but requires exponential size ROABP in another variable order. For example, the polynomial $(x_1 + y_1)(x_2 + y_2) \cdots (x_n + y_n)$ has a width-2 ROABP in the variable order $(x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$ (see Figure 2.2). On the other hand, any ROABP computing the same polynomial, in the variable order $(x_1, \ldots, x_n, y_1, \ldots, y_n)$, must have width at least 2^n (see [Nis91, GKST15] for lower bound techniques).

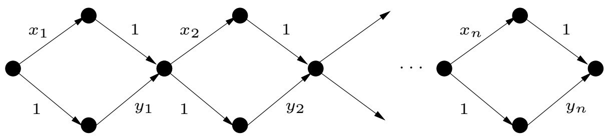

Figure 2.2: An ROABP computing the polynomial $(x_1 + y_1)(x_2 + y_2) \cdots (x_n + y_n)$.

Viewing $D_i(x_{\pi(i)}) \in \mathbb{F}^{w \times w}[x_{\pi(i)}]$ as a polynomial over the matrix algebra, we can write the polynomial computed by an ROABP as

$$C(\boldsymbol{x}) = U^{\mathsf{T}} D_1(x_{\pi(1)}) D_2(x_{\pi(2)}) \cdots D_n(x_{\pi(n)}) T.$$

An equivalent representation of an ROABP can be obtained by multiplying out U^{T} with D_1 and D_n with T.

$$C(\boldsymbol{x}) = D_1(x_{\pi(1)}) D_2(x_{\pi(2)}) \cdots D_n(x_{\pi(n)})$$

is said to be computed by a width-w ROABP if $D_1 \in \mathbb{F}^{1 \times w}[x_{\pi(1)}]$, $D_i \in \mathbb{F}^{w \times w}[x_{\pi(i)}]$ for $2 \leq i \leq n-1$ and $D_n \in \mathbb{F}^{w \times 1}[x_{\pi(n)}]$.

2.5.2 Sparse-factor ROABP

We call the ABP a *sparse-factor ROABP* if the edge weights in different layers are sparse polynomials in disjoint sets of variables. Formally, if there exists a partition of the variable set $\boldsymbol{x}$ into q sets $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_q\}$ such that $D_i \in \mathbb{F}^{w \times w}[\boldsymbol{x}_i]$, viewed as a polynomial over the matrix algebra, is a s-sparse polynomial for all $i \in [q]$, then the matrix product $U^{\mathsf{T}} \left(\prod_{i=1}^{q} D_i(\boldsymbol{x}_i)\right) T$ is called a *s-sparse-factor* ROABP. It is read once in the sense that any particular variable contributes to at most one edge on any path in the corresponding graph.

For designing the hitting-set we will consider the product $D(\boldsymbol{x}) = \prod_{i=1}^{q} D_i(\boldsymbol{x}_i)$, which is a polynomial over the matrix algebra $\mathbb{F}^{w \times w}$. As each D_i is a sparse polynomial, it has a depth-2 ($\Sigma\Pi$) circuit over $\mathbb{F}^{w \times w}$. Thus, the polynomial $D(\boldsymbol{x})$ has a depth-3 ($\Pi\Sigma\Pi$) circuit over $\mathbb{F}^{w \times w}$.

2.5.3 Commutative ROABP

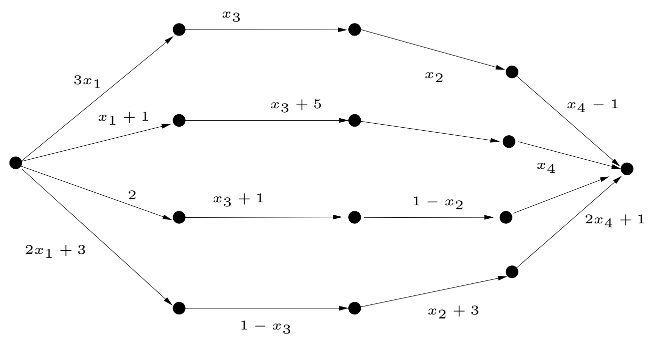

Figure 2.3: A commutative ROABP

An ROABP $U^{\mathsf{T}} \left(\prod_{i=1}^{q} D_i\right) T$ is a commutative ROABP, if all D_is are polynomials over a commutative subalgebra of the matrix algebra. For example, if the coefficients in the polynomials D_is are all diagonal matrices (see Figure 2.3). Note that the order of the variables becomes insignificant for a commutative ROABP. A polynomial computed by a commutative ROABP can be computed by an ROABP in any variable order.

It is easy to see that ROABPs subsume set-multilinear circuits and diagonal circuits.

Reduction from set-multilinear circuits to ROABP: Now, we argue that set-multilinear circuits are subsumed by sparse-factor ROABP. For a set-multilinear circuit

$$C(\boldsymbol{x}) = \sum_{i=1}^{k} \prod_{j=1}^{q} \ell_{ij}(\boldsymbol{x}_j),$$

define $D_j \in \mathbb{F}^{k \times k}[\boldsymbol{x}_j]$ as a diagonal matrix with the (i,i)-th entry being $\ell_{ij}(\boldsymbol{x}_j)$ and define U and T to be all one vectors. Clearly,

$$C(\boldsymbol{x}) = U^{\mathsf{T}} \left(\prod_{j=1}^{q} D_j(\boldsymbol{x}_j) \right) T.$$

Note that $D_j(\boldsymbol{x}_j)$, as a polynomial over $\mathbb{F}^{k \times k}$, is a linear polynomial, and hence is sparse.

Along the same lines one can argue that sum of products of univariate polynomials are captured by ROABP. In that case D_is will be diagonal matrices with entries being univariate polynomials. As diagonal circuits can be reduced to sum of products of univariate polynomials, they are also subsumed by ROABP.

In both the cases the resulting ROABP is a commutative sparse-factor ROABP, as D_is are diagonal matrices.

Chapter 3

Hitting-sets via Basis Isolation

In this chapter, we give new hitting-sets for ROABPs using a technique called basis isolation. For an n-variate, individual degree d polynomial computed by a width-w ROABP, our time complexity is $(ndw)^{O(\log n)}$. Our hitting-set construction goes along the lines of the whitebox test by Raz and Shpilka [RS05]. Our proof strategy was inspired by the rank-concentration ideas of Agrawal, Saha and Saxena [ASS13], who gave a blackbox test for set-multilinear circuits. We first describe the ideas behind these results.

3.1 Previous Works

3.1.1 Whitebox Test for ROABP [RS05]

Here, we give an overview of the polynomial-time whitebox test of Raz and Shpilka. They actually present their results for non-commutative ABPs, which can also be applied to ROABPs. Our presentation is different from theirs. Recall that a polynomial $C(\boldsymbol{x})$ is computed by a width-w ROABP, if $C(\boldsymbol{x}) = U^{\mathsf{T}}(\prod_{i=1}^{d} D_i)T$ with $U, T \in \mathbb{F}^{w \times 1}$ and $D_i \in \mathbb{F}^{w \times w}[x_i]$. Raz and Shpilka [RS05] work with the product $D := \prod_{i=1}^{d} D_i$, which is a polynomial over the matrix algebra $\mathbb{F}^{w \times w}$, i.e., its coefficients are $w \times w$ matrices. Observe that for any monomial $\boldsymbol{x}^{\boldsymbol{a}}$,

$$\mathrm{coef}_C(\boldsymbol{x}^{\boldsymbol{a}}) = U^{\mathsf{T}} \, \mathrm{coef}_D(\boldsymbol{x}^{\boldsymbol{a}}) T.$$

Hence, if $C(\boldsymbol{x}) \neq 0$, then clearly, there exists a monomial $\boldsymbol{x}^{\boldsymbol{a}}$ such that $U^\top \operatorname{coef}_D(\boldsymbol{x}^{\boldsymbol{a}})T \neq 0$. To test this, one just needs to find a basis for the coefficient space of $D(\boldsymbol{x})$.

Claim 3.1. *Let $C(\boldsymbol{x}) = U^\top D(\boldsymbol{x})T$. Let $\{\boldsymbol{x}^{\boldsymbol{a}_1}, \boldsymbol{x}^{\boldsymbol{a}_2}, \dots \boldsymbol{x}^{\boldsymbol{a}_k}\}$ be a set of monomials whose coefficients span the coefficient space of D. Then,*

$$C(\boldsymbol{x}) = 0 \iff U^\top \operatorname{coef}_D(\boldsymbol{x}^{\boldsymbol{a}_j})T = 0, \forall j \in [k].$$

Proof. If $U^\top \operatorname{coef}_D(\boldsymbol{x}^{\boldsymbol{a}_j})T \neq 0$ for some $j \in [k]$, then clearly $C(\boldsymbol{x}) \neq 0$.

Now, suppose $U^\top \operatorname{coef}_D(\boldsymbol{x}^{\boldsymbol{a}_j})T = 0$ for each $j \in [k]$. For any monomial $\boldsymbol{x}^{\boldsymbol{a}}$, its coefficient in D can be written as

$$\operatorname{coef}_D(\boldsymbol{x}^{\boldsymbol{a}}) = \sum_{j=1}^{k} \alpha_j \operatorname{coef}_D(\boldsymbol{x}^{\boldsymbol{a}_j})$$

for some constants $\{\alpha_j\}_j$. Hence, $\operatorname{coef}_C(\boldsymbol{x}^{\boldsymbol{a}}) = U^\top \operatorname{coef}_D(\boldsymbol{x}^{\boldsymbol{a}})T = 0$ by linearity of matrix multiplication. Thus, $C(\boldsymbol{x}) = 0$. $\square$

Now, the goal is to compute a set of monomials, whose coefficients in D span all the coefficients in D. Let $k = w^2$, the dimension of the underlying algebra. To get the essential idea, we consider the toy case when $D = D_1(x_1)D_2(x_2)$. First compute a basis for the coefficients of D_1 and D_2, separately. This can be done easily, as there are only $d + 1$ coefficients in both the polynomials D_1 and D_2 (d is the individual degree). Let B_1 and B_2 be the set of monomials corresponding to the basis coefficients in D_1 and D_2 respectively. Then for any $a_1, a_2 \in [\![d]\!]$, one can write

$$\operatorname{coef}_{D_1}(x_1^{a_1}) \in \operatorname{span}\{\operatorname{coef}_{D_1}(x_1^{b_1}) \mid x_1^{b_1} \in B_1\}, \tag{3.1}$$

and

$$\operatorname{coef}_{D_2}(x_2^{a_2}) \in \operatorname{span}\{\operatorname{coef}_{D_2}(x_2^{b_2}) \mid x_2^{b_2} \in B_2\}. \tag{3.2}$$

By multiplying (3.1) and (3.2), we get

$$\operatorname{coef}_{D_1}(x_1^{a_1}) \operatorname{coef}_{D_2}(x_2^{a_2}) \in \operatorname{span}\{\operatorname{coef}_{D_1}(x_1^{b_1}) \operatorname{coef}_{D_2}(x_2^{b_2}) \mid x_1^{b_1} \in B_1,\ x_2^{b_2} \in B_2\}. \tag{3.3}$$

Its easy to see that $\operatorname{coef}_{D_1}(x_1^{a_1}) \operatorname{coef}_{D_2}(x_2^{a_2}) = \operatorname{coef}_D(x_1^{a_1} x_2^{a_2})$, for any $a_1, a_2 \in [\![d]\!]$.

Thus, (3.3) can be rewritten as, for any $a_1, a_2 \in [\![d]\!]$,

$$\operatorname{coef}_D(x_1^{a_1}x_2^{a_2}) \in \operatorname{span}\{\operatorname{coef}_D(x_1^{b_1}x_2^{b_2}) \mid x_1^{b_1} \in B_1,\ x_2^{b_2} \in B_2\}.$$

In other words, the coefficients corresponding to the monomial set $B := B_1 \times B_2$ span the coefficient space of D. Clearly, $|B| \leq k^2$. Now, find a basis among these k^2 coefficients.

This procedure can be repeatedly applied to find a basis for the coefficients of the polynomial $D_1 D_2 \cdots D_n$. At the i-th step, let us say we have computed the set of basis monomials B_1 for the polynomial $D_1 D_2 \cdots D_i$. Let B_2 be a set of basis monomials for D_{i+1}. To compute a basis for the polynomial $D_1 D_2 \cdots D_{i+1}$, we just need to consider the monomial set $B_1 \times B_2$, which is of size k^2. Thus, at each step we need to compute a basis for a set of k^2 coefficients, which can be done efficiently.

3.1.2 Blackbox Test for Set-multilinear Circuits [ASS13]

As seen in Section 2.5, set-multilinear circuits are a subclass of ROABP. Hence, the white-box test of Raz and Shpilka [RS05] applies to them as well. The first completely blackbox test for set-multilinear circuits was given by Agrawal, Saha and Saxena [ASS13]. Recall that a set-multilinear circuit is of the form $C(\boldsymbol{x}) = \sum_{i=1}^{k} \prod_{j=1}^{q} \ell_{ij}(\boldsymbol{x}_j)$, where $\boldsymbol{x}_j$s are disjoint sets of variables and ℓ_{ij}s are linear polynomials. For the polynomial $C(\boldsymbol{x})$, they consider the polynomial $D = D_1 D_2 \cdots D_q$ over $\mathbb{H}_k$, where D_j is the vector polynomial $(\ell_{ij})_i \in \mathbb{F}[\boldsymbol{x}_j]^k$. It is easy to see that

$$C = (1, 1, \dots, 1) \cdot D.$$

Like the test of Raz and Shpilka, here also the goal is to find a basis for the coefficients in D, but in a blackbox manner. That is, to find a set of points in $\mathbb{F}^n$ such that the evaluations of D on those points span the coefficient space of D.

For this, Agrawal, Saha and Saxena use the idea of rank concentration.

Definition 3.2. A polynomial $D(\boldsymbol{x})$ over an algebra is said to be ℓ-concentrated if its coefficients of $(< \ell)$-support monomials span all its coefficients.

It is easy to see that for an ℓ-concentrated multilinear polynomial, its evaluations, on the set of points $\{h \in \{0,1\}^n \mid \|h\|_1 < \ell\}$, span its coefficient space [ASS13]. The cardinality of this set is $n^{O(\ell)}$, which is efficient for a small ℓ. However, the polynomial D, corresponding to a set-multilinear circuit, need not have ℓ-concentration. For example, consider the polynomial $\mathbf{1} \cdot x_1 x_2 \cdots x_n$, where $\mathbf{1} = (1, 1, \ldots, 1) \in \mathbb{H}_k$.

To achieve low-support concentration in set-multilinear circuits, Agrawal, Saha and Saxena [ASS13] use a shift of variables, i.e., replacing x_i with $x_i + t_i$ for each i, where $\{t_i\}_i$ are constants. For example, the shifted polynomial $\mathbf{1}(x_1+1)(x_2+1)\cdots(x_n+1)$ is 1-concentrated. For a general polynomial $D(\boldsymbol{x})$ over any k-dimensional algebra, one can show that when t_is are taken as formal variables, $D(x_1+t_1, \ldots, x_n+t_n)$ is $O(\log k)$-concentrated over the field $\mathbb{F}(t_1, \ldots, t_n)$ (see [ASS13, FSS14, AGKS13]). By the same arguments, the shift $x_i = x_i + \phi(x_i)$, for a univariate map $\phi \colon \boldsymbol{x} \to \{t^j\}_j$, does the same job if ϕ maps all the monomials to distinct powers of t (concentration will be over the field $\mathbb{F}(t)$). As the number of all monomials is exponential, separating all of them is not feasible. Instead, [ASS13] apply this argument only to small degree polynomials.

For the polynomial $D = D_1(\boldsymbol{x}_1)D_2(\boldsymbol{x}_2)\cdots D_q(\boldsymbol{x}_q)$, [ASS13] construct a shift such that for any set $\{i_1, i_2, \ldots, i_\ell\} \subseteq [q]$, the polynomial $D_{i_1}D_{i_2}\cdots D_{i_\ell}$ is ℓ-concentrated after the shift (the product is commutative). They take $\ell = O(\log k)$. As the number of degree-ℓ monomials is small, the cost of their shift is small. Their next step is to show that this shift suffices for an ℓ-concentration in the original polynomial D. The argument is somewhat similar to the one made in the previous section. Consider the toy case when D_is are univariates and $D = D_1(x_1)D_2(x_2)\cdots D_{\ell+1}(x_{\ell+1})$. Suppose the polynomial $D_0 = D_1(x_1)D_2(x_2)\cdots D_\ell(x_\ell)$ has ℓ-concentration. That is,

$$\mathrm{coef}_{D_0}(x_1 x_2 \cdots x_\ell) \in \mathrm{span}\{\mathrm{coef}_{D_0}(\boldsymbol{x}^{\boldsymbol{a}}) \mid \mathrm{supp}(\boldsymbol{a}) < \ell\}.$$

Multiplying $\mathrm{coef}_{D_{\ell+1}}(x_{\ell+1})$ with this gives us

$$\mathrm{coef}_D(x_1 x_2 \cdots x_{\ell+1}) \in \mathrm{span}\{\mathrm{coef}_D(\boldsymbol{x}^{\boldsymbol{a}} x_{\ell+1}) \mid \mathrm{supp}(\boldsymbol{a}) < \ell\}.$$

This shows that the coefficient of the monomial $x_1 x_2 \cdots x_{\ell+1}$ is in the span of coefficients which have support $\leq \ell$. But the $(\leq \ell)$-support coefficients are already in the span of ($< \ell$)-support coefficients, since we assumed ℓ-concentration in any $D_{i_1} D_{i_2} \cdots D_{i_\ell}$. One gets that ($\leq \ell + 1$)-support coefficients are also in the span of ($< \ell$)-support coefficients. For the general case $D = D_1 D_2 \cdots D_q$, one can extend this argument to higher and higher support coefficients, up to support q (see Lemma 5.8, for a detailed proof).

This shows that to achieve low-support concentration in a set-multilinear polynomial, separating all the monomials is not necessary, just separating all the small degree monomials suffices. One can ask if there is a direct proof for this, without going through small degree sub-circuits (like $D_1 D_2 \cdots D_\ell$). In search of such a proof, we found that a shift by a map ϕ which *isolates a basis* among the coefficients of D, achieves low-support concentration in D (see [GKST15, Kor15], for a detailed proof). A univariate map ϕ maps all the monomials to a power of t, which we refer as the weight of the monomial. Isolating a basis means that there exists a basis among the coefficients such that any non-basis coefficient depends on strictly smaller weight coefficients. The map of [ASS13] does this for set-multilinear circuits.

In the next section, we show that a basis isolating map can be constructed for an ROABP with a quasi-polynomial cost. Although basis isolation was motivated from rank-concentration, a hitting-set can be directly obtained from a basis isolating map, without using rank-concentration.

Forbes, Saptharishi and Shpilka [FSS14] have also used the rank-concentration ideas to construct a hitting-set for ROABPs. But, their cost is $(nw)^{d \log n \log w}$, where d is the individual degree.

3.2 Hitting-set for ROABP via Basis Isolation

Following the idea of [ASS13] and [FSS14], we work with polynomials over an algebra. That is, for a polynomial computed by a width-w ROABP, $C(\boldsymbol{x}) = U^{\mathsf{T}}(\prod_{i=1}^{d} D_i)T$, we view the product $D := \prod_{i=1}^{d} D_i$ as a polynomial over the matrix algebra $\mathbb{F}^{w \times w}$. We can write the polynomial $C(\boldsymbol{x})$ as the dot product $R \cdot D$, where $R = UT^{\mathsf{T}}$. We essentially try to construct a basis for the coefficient space of $D(\boldsymbol{x})$ by evaluations of $D(\boldsymbol{x})$. Clearly, if

$C \neq 0$, then the dot product of R with at least one of these basis vectors will be nonzero. And thus, we get a hitting-set.

Instead of shifting the polynomial $D(\boldsymbol{x})$ or breaking it into sub-circuits, as was done in [ASS13] and [FSS14], we directly work with $D(\boldsymbol{x})$. For a polynomial in $\mathbb{F}[\boldsymbol{x}]$, a usual technique for PIT is to give a univariate monomial map for the variables, such that a monomial of the given polynomial is isolated (e.g., sparse PIT [KS01]). In other words, the minimum weight monomial in the polynomial is unique. Our approach can be seen as a generalization of this technique. We come up with a univariate map (or weight function) on the variables which can *isolate a basis* for the coefficients of the polynomial $D(\boldsymbol{x}) \in \mathbb{F}^{w \times w}[\boldsymbol{x}]$.

We present our results for polynomials over an arbitrary algebra. Let $\mathbb{A}_k$ be a k-dimensional algebra over the field $\mathbb{F}$. Let $D(\boldsymbol{x})$ be a polynomial in $\mathbb{A}_k[\boldsymbol{x}]$ with individual degree d. Let M denote the set of all monomials over the variable set $\boldsymbol{x}$ with individual degree $\leq d$.

Now, we will define a basis isolating weight assignment for a polynomial $D \in \mathbb{A}_k[\boldsymbol{x}]$, which will lead to a hitting-set for the polynomial $C \in \mathbb{F}[\boldsymbol{x}]$, where $C = R \cdot D$, for some $R \in \mathbb{A}_k$. Recall that any function $\mathrm{w} \colon \boldsymbol{x} \to \mathbb{N}$ can be naturally extended to the set of all monomials as follows: $\mathrm{w}(\prod_{i=1}^{n} x_i^{\gamma_i}) = \sum_{i=1}^{n} \gamma_i \mathrm{w}(x_i)$, for any $(\gamma_i)_i \in \mathbb{N}^n$. Note that if variable x_i is replaced with $t^{\mathrm{w}(x_i)}$ for each i, then any monomial m just becomes $t^{\mathrm{w}(m)}$.

Definition 3.3 (Basis Isolating Weight Assignment)**.** A weight function $\mathrm{w} \colon \boldsymbol{x} \to \mathbb{N}$ is called a basis isolating weight assignment for a polynomial $D(\boldsymbol{x}) \in \mathbb{A}_k[\boldsymbol{x}]$, if there exists a set of monomials $S \subseteq M$ ($k' := |S| \leq k$) whose coefficients form a basis for the coefficient space of $D(\boldsymbol{x})$, such that

- for any $m, m' \in S$, $\mathrm{w}(m) \neq \mathrm{w}(m')$ and
- for any monomial $m \in M \setminus S$,

$$\mathrm{coef}_D(m) \in \mathrm{span}\{\mathrm{coef}_D(m') \mid m' \in S,\ \mathrm{w}(m') < \mathrm{w}(m)\}.$$

The above definition is equivalent to saying that there exists a *unique minimum* weight basis (according to the weight function w) among the coefficients of D, and also that the basis monomials have distinct weights. We skip the easy proof for this equivalence, as we will not need it. Let us emphasize here that according to this definition there could be many monomials in $M \setminus S$ which have the same weight as a monomial m in S. The only requirement is that their coefficients should be linearly dependent on basis coefficients with weight *smaller* than $\mathrm{w}(m)$.

Note that a weight assignment which gives distinct weights to all the monomials is indeed a basis isolating weight assignment. However, it will involve exponentially large weights. To find an efficient weight assignment one must use some properties of the given circuit. First, we show how such a weight assignment would lead to a hitting-set. We will actually show that it isolates a monomial in $C(\boldsymbol{x})$.

Lemma 3.4. *Let* $\mathrm{w}\colon \boldsymbol{x} \to \mathbb{N}$ *be a basis isolating weight assignment for a polynomial* $D(\boldsymbol{x}) \in \mathbb{A}_k[\boldsymbol{x}]$. *Let* $C = R \cdot D$ *be a nonzero polynomial for some* $R \in \mathbb{A}_k$. *Then, after the substitution* $x_i = t^{\mathrm{w}(x_i)}$ *for each* $i \in [n]$, *the polynomial* C *remains nonzero, where* t *is an indeterminate.*

Proof. For any monomial $m \in M$, let $D_m \in \mathbb{A}_k$ denote the coefficient $\mathrm{coef}_D(m)$. It is easy to see that after the mentioned substitution, the new polynomial $C'(t)$ is equal to $\sum_{m \in M} (R \cdot D_m) t^{\mathrm{w}(m)}$.

Let us say that $S \subset M$ is the set of monomials whose coefficients form the isolated basis for D. According to the definition of a basis isolating weight assignment, for any monomial $m \in M \setminus S$,

$$D_m \in \mathrm{span}\{D_{m'} \mid m' \in S,\ \mathrm{w}(m') < \mathrm{w}(m)\}. \tag{3.4}$$

First, we claim that there exists a monomial $m' \in S$ such that $R \cdot D_{m'} \neq 0$. For the sake of contradiction, let us assume that for all $m' \in S$, $R \cdot D_{m'} = 0$. Taking the dot product with R on both sides of (3.4), we get that for any monomial $m \in M \setminus S$,

$$R \cdot D_m \in \mathrm{span}\{R \cdot D_{m'} \mid m' \in S,\ \mathrm{w}(m') < \mathrm{w}(m)\}.$$

Hence, $R \cdot D_m = 0$ for all $m \in M$. This means that $C(\boldsymbol{x}) = 0$, which contradicts our assumption.

Now, let m^* be the minimum weight monomial in S whose coefficient gives a nonzero dot product with R, i.e.,

$$m^* = \operatorname*{arg\,min}_{m \in S} \{\mathrm{w}(m) \mid R \cdot D_m \neq 0\}.$$

There is a unique such monomial in S because all the monomials in S have distinct weights.

We claim that $\mathrm{coef}_{C'}(t^{\mathrm{w}(m^*)}) \neq 0$ and hence $C'(t) \neq 0$. To see this, consider any monomial m, other than m^*, with $\mathrm{w}(m) = \mathrm{w}(m^*)$. The monomial m has to be in the set $M \setminus S$, as the monomials in S have distinct weights. From (3.4),

$$D_m \in \mathrm{span}\{D_{m'} \mid m' \in S,\ \mathrm{w}(m') < \mathrm{w}(m^*)\}.$$

Taking the dot product with R on both sides, we get

$$R \cdot D_m \in \mathrm{span}\{R \cdot D_{m'} \mid m' \in S,\ \mathrm{w}(m') < \mathrm{w}(m^*)\}.$$

But, by the choice of m^*, $R \cdot D_{m'} = 0$ for any $m' \in S$ with $\mathrm{w}(m') < \mathrm{w}(m^*)$. Hence, $R \cdot D_m = 0$ for any $m \neq m^*$ with $\mathrm{w}(m) = \mathrm{w}(m^*)$.

Thus, the coefficient $\mathrm{coef}_{C'}(t^{\mathrm{w}(m^*)})$ can be written as

$$\sum_{\substack{m \in M \\ \mathrm{w}(m) = \mathrm{w}(m^*)}} R \cdot D_m = R \cdot D_{m^*},$$

which, as we know, is nonzero. □

We continue to use C' and S as in the proofs of Lemma 3.4. To construct a hitting-set for $C'(t)$, we can try many possible field values for t. The number of such values needed will be the degree of $C'(t)$, which is at most $(nd \max_i \mathrm{w}(x_i))$. Hence, the cost of the hitting-set is dominated by the *cost of the weight function*, i.e., the maximum weight given to any variable and the time taken to construct the weight function.

3.2.1 Basis Isolation for ROABP

In the next step, we show that such a basis isolating weight assignment can indeed be found for a sparse-factor ROABP, but with a cost quasi-polynomial in the input size. First, we make the following observation that it suffices that the coefficients of the monomials not in S, linearly depend on any coefficients with strictly smaller weight, not necessarily coming from S.

Observation 3.5. *If, for a polynomial $D \in \mathbb{A}_k[\boldsymbol{x}]$, there exist a weight function $\mathrm{w}\colon \boldsymbol{x} \to \mathbb{N}$ and a set of monomials $S \subseteq M$ ($k' := |S| \le k$) such that for any monomial $m \in M \setminus S$,*

$$\mathrm{coef}_D(m) \in \mathrm{span}\{\mathrm{coef}_D(m') \mid m' \in M,\ \mathrm{w}(m') < \mathrm{w}(m)\},$$

then we can also conclude that for any monomial $m \in M \setminus S$,

$$\mathrm{coef}_D(m) \in \mathrm{span}\{\mathrm{coef}_D(m') \mid m' \in S,\ \mathrm{w}(m') < \mathrm{w}(m)\}.$$

Proof. We are given that for any monomial $m \in \overline{S} := M \setminus S$,

$$\mathrm{coef}_D(m) \in \mathrm{span}\{\mathrm{coef}_D(m') \mid m' \in M,\ \mathrm{w}(m') < \mathrm{w}(m)\}.$$

Any coefficient $\mathrm{coef}_D(m')$ on the right-hand side of this equation which corresponds to a monomial in $\overline{S}$ can be replaced with some other coefficients which have further smaller weight. If we keep doing this, we will be left with only the coefficients corresponding to the set S, because in each step we are getting smaller and smaller weight coefficients. $\square$

In our construction of the weight function, we will create the set $\overline{S} := M \setminus S$ incrementally; i.e., in each step we will make more coefficients depend on strictly smaller weight coefficients. Finally, we will be left with only k' (the rank of the coefficient space of D) coefficients in S. We present the result for an arbitrary k-dimensional algebra $\mathbb{A}_k$ instead of just the matrix algebra.

Lemma 3.6 (Weight Construction)**.** *Let $\boldsymbol{x}$ be given by a union of q disjoint sets of variables $\boldsymbol{x}_1 \sqcup \boldsymbol{x}_2 \sqcup \cdots \sqcup \boldsymbol{x}_q$, with $|\boldsymbol{x}| = n$. Given k, s, d, we can construct in time $(ksn \log d)^{O(\log q)}$ a collection of weight assignments with weights bounded by $(ksn \log d)^{O(\log q)}$*

such that for any polynomial $D(\boldsymbol{x}) = P_1(\boldsymbol{x}_1)P_2(\boldsymbol{x}_2)\cdots P_q(\boldsymbol{x}_q)$, where $P_i \in \mathbb{A}_k[\boldsymbol{x}_i]$ is a s-sparse, individual degree-d polynomial for each $i \in [q]$, one of the weight assignments is basis isolating.

Proof. Let $\ell = \log q$. In our construction, a weight function w will be a combination of $(\ell+1)$ different weight functions, say $(\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_\ell)$. For any two monomials m and m', we say $\mathrm{w}(m) < \mathrm{w}(m')$ if $(\mathrm{w}_0(m), \mathrm{w}_1(m), \ldots, \mathrm{w}_\ell(m)) < (\mathrm{w}_0(m'), \mathrm{w}_1(m'), \ldots, \mathrm{w}_\ell(m'))$ according to the lexicographic ordering. Here, the weight functions $(\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_\ell)$ have their precedence in decreasing order from left to right, i.e., w_0 has the highest precedence and w_ℓ has the lowest precedence. As mentioned earlier, we will build the set $\overline{S}$ (the set of monomials whose coefficients are in the span of strictly smaller weight coefficients than themselves) incrementally in $(\ell+1)$ steps, using weight function w_i in the $(i+1)$-th step.

Let $M_{0,1}, M_{0,2}, \ldots, M_{0,q}$ be the sets of monomials and $\mathcal{C}_{0,1}, \mathcal{C}_{0,2}, \ldots, \mathcal{C}_{0,q}$ be the sets of coefficients in the polynomials $P_1, P_2, \ldots, P_q$, respectively.

Notation. The product of two sets of monomials M_1 and M_2 is defined as $M_1 \times M_2 = \{m_1 m_2 \mid m_1 \in M_1,\ m_2 \in M_2\}$. The product of any two sets of coefficients $\mathcal{C}_1$ and $\mathcal{C}_2$ is defined as $\mathcal{C}_1 \times \mathcal{C}_2 = \{c_1 c_2 \mid c_1 \in \mathcal{C}_1,\ c_2 \in \mathcal{C}_2\}$.

The crucial property of the polynomial D is that the set of coefficients in D, say $\mathcal{C}_0$, is just the product $\mathcal{C}_{0,1} \times \mathcal{C}_{0,2} \times \cdots \times \mathcal{C}_{0,q}$. Similarly, the set of all the monomials in D, say M_0, can be viewed as the product $M_{0,1} \times M_{0,2} \times \cdots \times M_{0,q}$. Let $m := m_a m_{a+1} \cdots m_b$ be a monomial, where $1 \le a \le b \le q$ and $m_j \in M_{0,j}$ for $a \le j \le b$. Then D_m will denote the coefficient $\mathrm{coef}_{P_a}(m_a)\,\mathrm{coef}_{P_{a+1}}(m_{a+1}) \cdots \mathrm{coef}_{P_b}(m_b)$.

Iteration 0: Let us fix $\mathrm{w}_0 \colon \boldsymbol{x} \to \mathbb{N}$ to be a weight function on the variables which gives distinct weights to all the s monomials in $M_{0,i}$ for each $i \in [q]$. As w_0 assigns distinct weights to these monomials, so does the weight function w.

For each P_i we do the following:

- arrange the coefficients in $\mathcal{C}_{0,i}$ in an increasing order of their weight according to w (or, equivalently, according to w_0),
- choose a maximal set of linearly independent coefficients, in a greedy manner, going

from lower weights to higher weights.

The fact that the weight functions $\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_\ell$ are not defined yet does not matter because w_0 has the highest precedence. The total order given to the monomials in $M_{0,i}$ by w_0 is the same as that given by w, irrespective of what the functions $\mathrm{w}_1, \ldots, \mathrm{w}_\ell$ are chosen to be.

This gives us a basis for the coefficients of P_i, say $\mathcal{C}'_{0,i}$. Let $M'_{0,i}$ denote the monomials in P_i corresponding to these basis coefficients. From the construction of the basis, it follows that for any monomial $m \in M_{0,i} \setminus M'_{0,i}$,

$$D_m \in \mathrm{span}\{D_{m'} \mid m' \in M'_{0,i},\ \mathrm{w}(m') < \mathrm{w}(m)\}. \tag{3.5}$$

Now, consider any monomial $m \in M_0$ which is not present in the set $M'_0 := M'_{0,1} \times M'_{0,2} \times \cdots \times M'_{0,q}$. Let $m = m_1 m_2 \cdots m_q$, where $m_i \in M_{0,i}$ for all $i \in [q]$. We know that for at least one $j \in [q]$, $m_j \in M_{0,j} \setminus M'_{0,j}$. Then using (3.5) we can write the following about $D_m = D_{m_1} D_{m_2} \cdots D_{m_q}$:

$$D_m \in \mathrm{span}\{D_{m_1} \cdots D_{m_{j-1}} D_{m'_j} D_{m_{j+1}} \cdots D_{m_q} \mid m'_j \in M'_{0,j},\ \mathrm{w}(m'_j) < \mathrm{w}(m_j)\}.$$

This holds because the algebra product is bilinear. Equivalently, for any monomial $m \in M_0 \setminus M'_0$,

$$D_m \in \mathrm{span}\{D_{m'} \mid m' \in M_0,\ \mathrm{w}(m') < \mathrm{w}(m)\}.$$

This is true because

$$\mathrm{w}(m_1) + \cdots + \mathrm{w}(m'_j) + \cdots + \mathrm{w}(m_q) < \mathrm{w}(m_1) + \cdots + \mathrm{w}(m_j) + \cdots + \mathrm{w}(m_q) = \mathrm{w}(m).$$

Hence, all the monomials in $M_0 \setminus M'_0$ can be put into $\overline{S}$; i.e., their corresponding coefficients depend on strictly smaller weight coefficients.

Iteration 1: Now, let us consider monomials in the set $M'_0 = M'_{0,1} \times M'_{0,2} \times \cdots \times M'_{0,q}$. Let the corresponding set of coefficients be $\mathcal{C}'_0 := \mathcal{C}'_{0,1} \times \mathcal{C}'_{0,2} \times \cdots \times \mathcal{C}'_{0,q}$. Since the underlying algebra $\mathbb{A}_k$ has dimension at most k and the coefficients in $\mathcal{C}'_{0,i}$ form a basis for $\mathcal{C}_{0,i}$, $|M'_{0,i}| \leq k$ for all $i \in [q]$. In the above product, let us make $q/2$ disjoint pairs of consecutive terms and for each pair, multiply the two terms in it. Putting it formally, let us define $\mathcal{C}_{1,j}$ to be the product $\mathcal{C}'_{0,2j-1} \times \mathcal{C}'_{0,2j}$ and similarly $M_{1,j} := M'_{0,2j-1} \times M'_{0,2j}$ for all $j \in [q/2]$ (if q is odd, we can make it even by multiplying the identity element of $\mathbb{A}_k$ in the end). Now, let $\mathcal{C}_1 := \mathcal{C}'_0 = \mathcal{C}_{1,1} \times \mathcal{C}_{1,2} \times \cdots \times \mathcal{C}_{1,q_1}$ and $M_1 := M'_0 = M_{1,1} \times M_{1,2} \times \cdots \times M_{1,q_1}$, where $q_1 := q/2$. For any $i \in [q_1]$, $M_{1,i}$ has at most k^2 monomials.

Now we fix the weight function $\mathrm{w}_1 \colon \boldsymbol{x} \to \mathbb{N}$ such that it gives distinct weights to all the monomials in $M_{1,i}$ for each $i \in [q_1]$. As w_1 separates these monomials, so does the weight function w. Now we repeat the same procedure of constructing a basis in a greedy manner for $\mathcal{C}_{1,i}$ according to the weight function w for each $i \in [q_1]$. Let the basis coefficients for $\mathcal{C}_{1,i}$ be $\mathcal{C}'_{1,i}$ and the corresponding monomials be $M'_{1,i}$.

As argued before, any coefficient in $\mathcal{C}_1$ which is outside the set $\mathcal{C}'_1 := \mathcal{C}'_{1,1} \times \mathcal{C}'_{1,2} \times \cdots \times \mathcal{C}'_{1,q_1}$ is in the span of strictly smaller weight (than itself) coefficients. Thus, we can also put the corresponding monomials $M_1 \setminus M'_1$ in $\overline{S}$, where $M'_1 := M'_{1,1} \times M'_{1,2} \times \cdots \times M'_{1,q_1}$.

Iteration r: We keep repeating the same procedure for $(\ell+1)$ rounds. After round r, say, the set of monomials we are left with is given by the product $M'_{r-1} = M'_{r-1,1} \times M'_{r-1,2} \times \cdots \times M'_{r-1,q_{r-1}}$, where $q_{r-1} = q/2^{r-1}$. Here, $M_{r-1,i}$ has at most k monomials for each $i \in [q_{r-1}]$. In the above product, we make $q_{r-1}/2$ disjoint pairs of consecutive terms and multiply the two terms in each pair. Let us say we get $M_r := M'_{r-1} = M_{r,1} \times M_{r,2} \times \cdots \times M_{r,q_r}$, where $q_r = q_{r-1}/2$. Say, the corresponding set of coefficients is given by $\mathcal{C}_r = \mathcal{C}_{r,1} \times \mathcal{C}_{r,2} \times \cdots \times \mathcal{C}_{r,q_r}$. Note that $|M_{r,i}| \leq k^2$ for each $i \in [q_r]$.

We fix the weight function w_r such that it gives distinct weights to all the monomials in the set $M_{r,i}$ for each $i \in [q_r]$. We once again mention that fixing of w_r does not affect the greedy basis constructed in earlier rounds and hence the monomials which were put in the set $\overline{S}$, because w_r has less precedence than any $\mathrm{w}_{r'}$ for $r' < r$.

For each $\mathcal{C}_{r,i}$, we construct a basis in a greedy manner going from lower weight to higher weight (according to the weight function w). Let this set of basis coefficients be $\mathcal{C}'_{r,i}$ and the corresponding monomials be $M'_{r,i}$ for each $i \in [q_r]$. Let $\mathcal{C}'_r := \mathcal{C}'_{r,1} \times \mathcal{C}'_{r,2} \times \cdots \times \mathcal{C}'_{r,q_r}$ and $M'_r := M'_{r,1} \times M'_{r,2} \times \cdots \times M'_{r,q_r}$. Arguing similarly as before we can say that each coefficient in $\mathcal{C}_{r,i} \setminus \mathcal{C}'_{r,i}$ is in the span of strictly smaller weight coefficients (from $\mathcal{C}'_{r,i}$) than itself. Hence, the same can be said about any coefficient in the set $\mathcal{C}_r \setminus \mathcal{C}'_r$. So, all the monomials in the set $M_r \setminus M'_r$ can be put into $\overline{S}$. Now, we are left with monomials $M'_r = M'_{r,1} \times M'_{r,2} \times \cdots \times M'_{r,q_r}$ for the next round.

Iteration ℓ: As in each round the number of terms in the product gets halved, after ℓ rounds we will be left with just one term, i.e., $M_\ell = M'_{\ell-1,1} \times M'_{\ell-1,2} = M_{\ell,1}$. Now, we will fix the function w_ℓ which separates all the monomials in $M_{\ell,1}$. By arguments similar to those above, we will finally be left with at most k' monomials in S, which will all have distinct weights. It is clear that for every monomial in $\overline{S}$, its coefficient will be in the span of strictly smaller weight coefficients than itself.

Now, let us look at the cost of this weight function. In the first round, w_0 needs to separate at most $O(qs^2)$ many pairs of monomials. For each $1 \leq r \leq \ell$, w_r needs to separate at most $O(qk^4)$ many pairs of monomials. From Lemma 2.3, to construct w_r, for any $0 \leq r \leq \ell$, one needs to try $\mathsf{poly}(k, s, n, \log d)$ many weight functions each having highest weight $\mathsf{poly}(k, s, n, \log d)$ (as q is bounded by n). To get the correct combination of the weight functions $(\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_\ell)$ we need to try all possible combinations of these polynomially many choices for each w_r. This gives us a collection of $(ksn \log d)^{O(\ell)}$ weight functions.

To combine these weight functions we can choose a large enough number B (greater than the highest weight a monomial can get in any of the weight functions), and define $\mathrm{w} := \mathrm{w}_0 B^\ell + \mathrm{w}_1 B^{\ell-1} + \cdots + \mathrm{w}_\ell$. The choice of B ensures that the different weight functions cannot interfere with each other, and they also get the desired precedence order.

The highest weight a monomial can get from the weight function w is $(ksn \log d)^{O(\ell)}$. Thus, the cost of w remains $(ksn \log d)^{O(\ell)}$.

$\square$

Combining Lemma 3.6 with Observation 3.5 and Lemma 3.4, we can get a hitting-set for ROABP.

Theorem 3.7. *Let $C(\boldsymbol{x})$ be an n-variate polynomial computed by a width-w, s-sparse-factor ROABP, with* indv-deg$(C) \leq d$. *Then there is a $(wsnd)^{O(\log n)}$-time hitting-set for*

$C(\boldsymbol{x})$.

Proof. As mentioned earlier, $C(\boldsymbol{x})$ can be written as $R \cdot D(\boldsymbol{x})$ for some $R \in \mathbb{F}^{w \times w}$, where $D(\boldsymbol{x}) \in \mathbb{F}^{w \times w}[\boldsymbol{x}]$. The underlying matrix algebra $\mathbb{F}^{w \times w}$ has dimension $k = w^2$. The hitting-set size will be dominated by the cost of the weight function constructed in Lemma 3.6. After the univariate substitution $x_i \mapsto t^{\mathrm{w}(x_i)}$, the polynomial $C(\boldsymbol{x})$ becomes a univariate polynomial with degree $\leq nd \max_i \mathrm{w}(x_i)$. This will be bounded by $(wsnd)^{O(\log n)}$ (the parameter q in Lemma 3.6, i.e., the number of layers in the ROABP, is bounded by n). The final hitting-set will be to plug-in $(wsnd)^{O(\log n)}$ distinct field values for t. Thus, the hitting-set size will be $(wsnd)^{O(\log n)}$. $\square$

Field Size: After substituting $x_i = t^{\mathrm{w}(x_i)}$ in the polynomial $C(\boldsymbol{x})$ for each $i \in [n]$, where w comes from Lemma 3.6, it becomes a nonzero univariate polynomial with quasi-polynomial degree. The hitting-set for this univariate polynomial will require the field size to be quasi-polynomial. If we want to find a hitting-set with only a polynomial size field, then we can do the following: do not combine the different weight functions $(\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_\ell)$ in the proof of Lemma 3.6. Instead, substitute $x_i = t_0^{\mathrm{w}_0(x_i)} t_1^{\mathrm{w}_1(x_i)} \cdots t_\ell^{\mathrm{w}_\ell(x_i)}$ for each i, where $\{t_j\}_j$ are distinct variables. By similar arguments, one can show that the polynomial remains nonzero after this substitution. As the degree in each variable t_j is polynomially bounded, a hitting-set of the form $H^{\ell+1}$ suffices, where $H \subseteq \mathbb{F}$ has polynomially bounded size (Lemma 2.2).

3.3 Basis Isolation in Diagonal Circuits

In this section, we apply the basis isolation technique to diagonal circuits to get a hitting-set. Recall that a diagonal circuit is a circuit of the form $\sum_{i=1}^{k} (\ell_i(\boldsymbol{x}))^{d_i}$, where $\ell_i(\boldsymbol{x})$ is a linear polynomial for each i. Saxena [Sax08] showed that any polynomial computed by a diagonal circuit can also be computed by $\mathsf{poly}(k, n, d)$-size ROABP, where d is the degree of the polynomial (see Lemma 2.4). Thus, there is a $(nkd)^{O(\log n)}$-time hitting-set for diagonal circuits. However, there are better results known for diagonal circuits. It is known that any polynomial computed by a diagonal circuit always has a low-support monomial with a nonzero coefficient. Agrawal, Saha and Saxena [ASS13] and Forbes and Shpilka [FS13b] prove this result using different techniques and with different parameters. Forbes, Saptharishi and Shpilka [FSS14] use this fact to give a $(nkd)^{O(\log\log k)}$-time hitting-set for diagonal circuits. Here, we give a new hitting-set for diagonal circuits using the basis isolation technique (without a reduction to ROABP). Our time complexity is $(nkd)^{\log d}$, which is incomparable to the previous results. However, our hitting-set works only when the field characteristic is zero or greater than the degree of the polynomial. Getting a polynomial-time hitting-set for diagonal circuits remains an open question.

Before applying the basis isolation technique, we first reduce the PIT question to homogeneous diagonal circuits. A homogeneous diagonal circuit is given by the form $\sum_{i=1}^{k}(\sum_{j=1}^{n} a_{ij}x_j)^d$. Given a power of a linear polynomial $(a_0 + \sum_{j=1}^{n} a_j x_j)^d$ one can expand it to write it as sum of homogeneous parts.

$$\left(a_0 + \sum_{j=1}^{n} a_j x_j\right)^d = a_0^d + d a_0^{d-1}\left(\sum_{j=1}^{n} a_j x_j\right) + \cdots + \left(\sum_{j=1}^{n} a_j x_j\right)^d .$$

Thus, one can write a diagonal circuit f as a sum of homogeneous diagonal circuits with different degrees, say, $f = f_0 + f_1 + \cdots + f_d$. To reduce the PIT question simply replace each variable x_i with $x_i t$. This separates different degree terms, i.e., $f(\boldsymbol{x}t) = f_0(\boldsymbol{x}) + f_1(\boldsymbol{x})t + \cdots + f_d(\boldsymbol{x})t^d$. Clearly, a hitting-set for homogeneous diagonal circuits, together with $d+1$ distinct values for t, will work as a hitting-set for $f(\boldsymbol{x}t)$.

3.3.1 Basis Isolation in Homogeneous Diagonal Circuits

Now, we move on to show basis isolation for homogeneous diagonal circuits. For that we consider the polynomial over Hadamard algebra. That is, we write

$$\sum_{i=1}^{k}\left(\sum_{j=1}^{n} a_{ij}x_j\right)^d = (1, 1, \ldots, 1)\cdot\left(\sum_{j=1}^{n} v_j x_j\right)^d ,$$

where $v_i = (a_{ij})_i$ is a k-dimensional vector for each j. Like Lemma 3.6, we will create the set $\overline{S} := M \setminus S$ incrementally in $\log d$ rounds. Finally, we will get the set S of monomials, whose coefficients form the isolated basis.

Lemma 3.8. *Let $D(\boldsymbol{x}) = (\sum_{j=1}^{n} v_j x_j)^d$ be a polynomial over the k-dimensional Hadamard algebra $\mathbb{H}_k(\mathbb{F})$, where $\operatorname{char}(\mathbb{F})$ is either zero or greater than d. Then we can construct a basis isolating weight assignment for $D(\boldsymbol{x})$ with cost $(ndk)^{O(\log d)}$.*

Proof. In the polynomial $D(\boldsymbol{x}) = (\sum_{j=1}^{n} v_j x_j)^d$, the coefficient of any monomial is given by

$$\operatorname{coef}_D(x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}) = \binom{d}{a_1\ a_2\ \cdots\ a_n} v_1^{a_1} v_2^{a_2} \cdots v_n^{a_n}, \tag{3.6}$$

where $\binom{d}{a_1\ a_2\ \cdots\ a_n}$ is the number of ways of choosing an ordered collection of n subsets of $[d]$ with their sizes being $a_1, a_2, \ldots, a_n$. It is given by the following:

$$\binom{d}{a_1\ a_2\ \cdots\ a_n} = \binom{d}{a_1}\binom{d-a_1}{a_2}\cdots\binom{d-a_1-a_2\cdots-a_{n-1}}{a_n}.$$

Note that the number $\binom{d}{a_1\ a_2\ \cdots\ a_n}$ is nonzero in the field $\mathbb{F}$, as $\operatorname{char}(\mathbb{F}) = 0$ or $\operatorname{char}(\mathbb{F}) > d$. This will be important in the proof when we have a division by this number.

Let $\ell := \lfloor \log d \rfloor$. In our construction, the final weight function w will be a combination of $(\ell+1)$ different weight functions, say $(\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_\ell)$. For any two monomials m and m', we say $\mathrm{w}(m) < \mathrm{w}(m')$ if $(\mathrm{w}_0(m), \mathrm{w}_1(m), \ldots, \mathrm{w}_\ell(m)) < (\mathrm{w}_0(m'), \mathrm{w}_1(m'), \ldots, \mathrm{w}_\ell(m'))$ according to the lexicographic ordering. Here, the weight functions $(\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_\ell)$ have their precedence in decreasing order from left to right, i.e., w_0 has the highest precedence and w_ℓ has the lowest precedence. Let $d_i = \lfloor d/2^{\ell-i} \rfloor$ for each $0 \le i \le \ell$. Our construction will involve $\ell+1$ iterations. In the i-th iteration, we will fix the weight function w_i such that the combined weight function $(\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_i)$ will be basis isolating for the polynomial $D_i := (\sum_{j=1}^{n} v_j x_j)^{d_i}$ for each $0 \le i \le \ell$. Let M_i and $\mathcal{C}_i$ denote the set of monomials and coefficients of D_i, respectively.

For any two set S_1, S_2 of monomials or coefficient vectors, $S_1 \times S_2$ is defined to be $\{s_1 s_2 \mid s_1 \in S_1,\ s_2 \in S_2\}$. Similarly, $S_1^2 := \{s_1 s_2 \mid s_1, s_2 \in S_1\}$.

Iteration 0*:* Define the function w_0 as $\mathrm{w}_0(x_i) := i$ for each $i \in [n]$. w_0 gives different weights to the monomials $\{x_1, x_2, \ldots, x_n\}$ and thus is basis isolating for the polynomial $D_0 = (\sum_{j=1}^{n} v_j x_j)$. From the set of coefficients $\mathcal{C}_0 = \{v_1, v_2, \ldots v_n\}$, choose a maximal set of linearly independent coefficients, in a greedy manner, going from lower weights to higher weights. This gives us a basis for the coefficients in $\mathcal{C}_0$, say $\mathcal{C}_0'$. Let the corresponding set of monomials be M_0'. Clearly, w_0 is basis isolating for D_0 with the isolated basis being $\mathcal{C}_0'$.

Iteration i: Let us assume the weight function $\mathrm{w}^* := (\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_{i-1})$ is basis isolating for the polynomial D_{i-1}. Let the set of coefficients forming the isolated basis in D_{i-1} be $\mathcal{C}_{i-1}'$ and the corresponding set of monomials be M_{i-1}'. Clearly, $|\mathcal{C}_{i-1}'| = |M_{i-1}'| \leq k$. We need to consider two cases depending on whether d_i is even or odd.

Case 1: Let us take first case when $d_i = 2d_{i-1}$ and $D_i = D_{i-1}^2$. Clearly the set of coefficients $\mathcal{C}_i = \mathcal{C}_{i-1}^2$. For any monomial $m \in M_i \setminus (M_{i-1}')^2$, we can write $m = m_1 m_2$ with $m_1, m_2 \in M_{i-1}$ such that either m_1 or m_2 lies in $M_{i-1} \setminus M_{i-1}'$. Without loss of generality, let m_1 be that monomial.

From the definition of an isolated basis for D_{i-1}, it follows that

$$\mathrm{coef}_{D_{i-1}}(m_1) \in \mathrm{span}\{\mathrm{coef}_{D_{i-1}}(m') \mid m' \in M_{i-1}',\ \mathrm{w}^*(m') < \mathrm{w}^*(m_1)\}. \tag{3.7}$$

From Equation (3.6), it is easy to see that for any monomials $m_1 = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$ and $m_2 = x_1^{b_1} x_2^{b_2} \cdots x_n^{b_n}$ in M_{i-1},

$$\mathrm{coef}_{D_i}(m_1 m_2) = \alpha \, \mathrm{coef}_{D_{i-1}}(m_1) \, \mathrm{coef}_{D_{i-1}}(m_2),$$

where $\alpha = \binom{d_i}{a_1+b_1\ a_2+b_2\ \cdots\ a_n+b_n} / \left(\binom{d_{i-1}}{a_1\ a_2\ \cdots\ a_n} \binom{d_{i-1}}{b_1\ b_2\ \cdots\ b_n} \right)$. Note that α is well defined as the binomial coefficients are all nonzero in $\mathbb{F}$.

Thus, on multiplying $\alpha \, \mathrm{coef}_{D_{i-1}}(m_2)$ with (3.7), we get

$$\mathrm{coef}_{D_i}(m) \in \mathrm{span}\{\mathrm{coef}_{D_i}(m' m_2) \mid m' \in M_{i-1}',\ \mathrm{w}^*(m') < \mathrm{w}^*(m_1)\}.$$

Since $m' m_2$ is a monomial in M_i, we can write

$$\mathrm{coef}_{D_i}(m) \in \mathrm{span}\{\mathrm{coef}_{D_i}(m') \mid m' \in M_i,\ \mathrm{w}^*(m') < \mathrm{w}^*(m)\}.$$

This shows that for any monomial $m \in M_i \setminus (M_{i-1}')^2$, its coefficient depends on strictly smaller weight coefficients. As this is true with respect to the weight function $\mathrm{w}^* = (\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_{i-1})$, it will also be true with respect to $(\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_i)$, whatever the function w_i we fix.

Now, we are only left with monomials in $(M'_{i-1})^2$. To isolate a basis among the coefficient set $(\mathcal{C}'_{i-1})^2$, we simply fix the function w_i such that it separates all the monomials in $(M'_{i-1})^2$, which has at most k^2 monomials. Thus, the weight function $(\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_i)$ becomes basis isolating for D_i.

Case 2: The case when d_i is odd, is similar to Case 1. We have $d_i = 2d_{i-1} + 1$ and $D_i = D_{i-1}^2 D_0$. With arguments similar to those in Case 1, we can show that for any monomial $m \in M_i \setminus ((M'_{i-1})^2 M'_0)$, its coefficient depends on strictly smaller weight coefficients. Then we can fix w_i to separate all the monomials in $(M'_{i-1})^2 M'_0$, which has at most k^3 monomials.

Clearly, after iteration ℓ we get the weight function $(\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_\ell)$ which is basis isolating for D.

Time Complexity: Now, let us look at the cost of this weight function. For each $0 \leq i \leq \ell$, w_i needs to separate at most $O(k^6)$ many pairs of monomials. From Lemma 2.3, to construct w_i one needs to try $\mathsf{poly}(k, n, d)$-many weight functions each having highest weight $\mathsf{poly}(k, n, d)$. To get the correct combination of the weight functions $(\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_\ell)$ we need to try all possible combinations of these polynomially many choices for each w_i. Thus, we have to try $(knd)^{O(\ell)}$ many combinations. To get the final weight function w, we an combine these weight functions, as was done in Lemma 3.6, by choosing a large enough base.

The highest weight a monomial can get from the weight function w is $(knd)^{O(\log d)}$. Thus, the cost of w remains $(knd)^{O(\log d)}$.

□

By combining Lemma 3.8 with Lemma 3.4, we get a $(knd)^{O(\log d)}$-time hitting-set for homogeneous diagonal circuits. As discussed earlier this will give a hitting-set for general diagonal circuits, with only a polynomial overhead.

Theorem 3.9. *Let $\mathbb{F}$ be a field with $\mathrm{char}(\mathbb{F}) = 0$ or $\mathrm{char}(\mathbb{F}) > d$. Then for a polynomial computed by a diagonal circuit $\sum_{i=1}^{k}(a_{i0} + \sum_{j=1}^{n} a_{ij}x_j)^{d_i}$ over the field $\mathbb{F}$, there is a $(knd)^{O(\log d)}$-time hitting-set, where $d = \max\{d_i\}_i$.*

3.4 Discussion

Any polynomial-time hitting-set for ROABP or even for diagonal circuits is not known till now. One possible approach for this would be to do a basis isolation with only a *polynomially* large weight assignment. Also, our technique of finding a basis isolating weight assignment seems general. It needs to be explored, for what other classes can it be applied. In particular, can it be used to solve depth-3 multilinear circuits?

With our result on ROABP we matched the complexity of the unknown-order case with the known-order case. One can ask whether we can find a similar result in the Boolean setting, i.e., get a pseudorandom generator for unknown-order ROBP with seed length same as the known-order case. Currently there is big gap in the seed lengths of these two cases ($O(\sqrt{n})$ versus $O(\log^2 n)$).

Chapter 4

Testing Zeroness of an XOR of Read-once Ordered Branching Programs

The previous chapter solves PIT for polynomials computed by read-once oblivious arithmetic branching programs (ROABP). The next natural question would be to solve PIT for a sum of two ROABPs (possibly, in different variable orders). We gave a polynomial time whitebox test and a quasi-polynomial time blackbox test for this question in [GKST15]. We also extended it to a sum of constantly many ROABPs. The idea was inspired by a similar question in the Boolean setting: testing equivalence of two read-once ordered branching programs (ROBP). ROBP (defined below) is a model analogous to ROABP which computes Boolean functions. Note that testing zeroness of a sum of two ROABPs is same as testing equivalence of the two given ROABPs.

Bryant [Bry86] defined the notion of ROBP (they call it OBDD) and gave its basic properties. Later Savický and Wegener [SW97] considered the problem of testing whether two given ROBPs (possibly, in different variable orders) compute the same Boolean function. They gave a polynomial time algorithm for this. Like in the arithmetic setting, we generalize their result to constantly many ROBPs. That is, we give a polynomial time algorithm to test whether an XOR of constantly many ROBPs is the zero function (unsatisfiable). In this chapter, we first describe the idea of Savický and Wegener [SW97] for two ROBPs and then present our algorithm for an XOR of c ROBPs. The time complexity of our algorithm is $\mathsf{poly}(n^c w^{2^c})$. For the PIT results on sum of ROABPs, see Korwar's thesis [Kor15].

4.1 Read-once Ordered Branching Programs

We start by defining ROBPs which are also known as ordered binary decision diagrams (OBDD).

Definition 4.1. An n-variate ROBP is a directed acyclic layered graph with $n + 1$ layers of vertices such that

- The first layer contains a single node s, called the starting node.
- The last layer contains two nodes t_1 and t_0, called accepting and rejecting nodes, respectively.
- Each node in the first n layers has two outgoing edges, each going to a vertex in the next layer. One is labeled with 0 and another with 1.
- The i-th layer of vertices has an associated variable $x_{\pi(i)}$ for each $i \in [n]$, where π is a permutation on $[n]$.

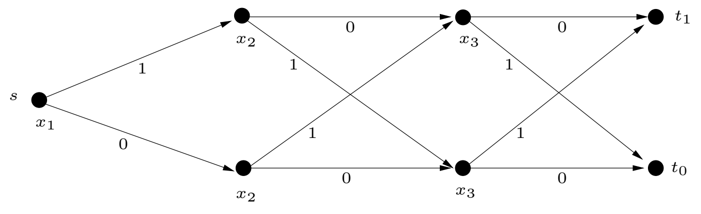

Figure 4.1: An ROBP computing the function $x_1 \oplus x_2 \oplus x_3$.

Figure 4.1 illustrates an ROBP. For any given assignment $(a_1, a_2, \ldots, a_n) \in \{0,1\}^n$ of the variables $\boldsymbol{x}$, one can construct a path in the ROBP starting from s: at a node in the

i-th layer take the edge labeled with $a_{\pi(i)}$ for each $i \in [n]$. It is easy to see that for each $\boldsymbol{a} \in \{0,1\}^n$, one gets a unique path ending in either the accepting node or the rejecting node. The ROBP is said to compute a function $f\colon \{0,1\}^n \to \{0,1\}$ such that for any $\boldsymbol{a} \in \{0,1\}^n$, $f(\boldsymbol{a}) = 1$ if the path corresponding to $\boldsymbol{a}$ ends in the accepting state and $f(\boldsymbol{a}) = 0$ otherwise. Similarly, any node v in the first n layers can be said to compute a Boolean function which is defined by the paths starting from v and ending in t_1 or t_0. A node in the i-th layer computes a function on variables $\{x_{\pi(i)}, x_{\pi(i+1)}, \ldots, x_{\pi(n)}\}$.

$(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$ is said to be the variable order of the ROBP. The maximum number of vertices in any layer is called the *width* of the ROBP. A width-w ROBP can be seen as a machine with w states, reading the variables in a fixed order. Next, we discuss a characterization of all the Boolean functions which can be computed by a ROBP in a particular variable order, given by Bryant [Bry86].

4.1.1 Characterization of an ROBP

The characterization of an ROBP is given in terms of partial evaluations or restrictions of the function. For ease of notation we will consider the standard variable order $(x_1, x_2, \ldots, x_n)$. For any function f and for any $\boldsymbol{a} = (a_1, a_2, \ldots, a_i) \in \{0,1\}^i$, $f_{\boldsymbol{a}}$ will denote the restriction of f on first i variables, i.e.,

$$f_{\boldsymbol{a}} = f(x_1 = a_1, x_2 = a_2, \ldots, x_i = a_i, x_{i+1}, \ldots, x_n)$$

for any $i \in [n]$. Clearly, $f_{\boldsymbol{a}}$ is a function on the variables $\{x_{i+1}, \ldots, x_n\}$. The characterization of the ROBP relates its width with the number of distinct functions among all the restrictions of f for a fixed i.

Lemma 4.2 ([Bry86])**.** *A Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ is computed by an ROBP of width-w in variable order $(x_1, x_2, \ldots, x_n)$ if and only if*

$$|\{f_{\boldsymbol{a}} \mid \boldsymbol{a} \in \{0,1\}^i\}| \le w$$

for each $i \in [n]$.

Proof sketch. ($\Longrightarrow$) The forward direction is simple. Let $\boldsymbol{a} \in \{0,1\}^i$ for some $i \in [n]$.

For a given ROBP computing f, take the unique path in the ROBP corresponding to the assignment $\boldsymbol{a}$. Let us say the path ends in a node v in the $(i+1)$-th layer. It is easy to see that the $f_{\boldsymbol{a}}$ is nothing but the function computed by the node v. As there are at most w nodes in any layer, the number of distinct functions in the set $\{f_{\boldsymbol{a}} \mid \boldsymbol{a} \in \{0,1\}^i\}$ is bounded by w.

($\Longleftarrow$) Now we prove the other direction. Let f be such that

$$|\{f_{\boldsymbol{a}} \mid \boldsymbol{a} \in \{0,1\}^i\}| \leq w$$

for each $i \in [n]$. We will build a width-w ROBP for f iteratively. Put the start node s. In the second layer put two notes v_0 and v_1 and draw edges (s, v_0) and (s, v_1) with labels 0 and 1 respectively. v_0 and v_1 are supposed to compute the functions f_0 and f_1 respectively. The same way, one can make the third layer with four nodes which are supposed to compute functions $f_{0,0}, f_{0,1}, f_{1,0}, f_{1,1}$. One can keep doing this, but the number of nodes grow exponentially with each layer. The idea is to reduce the number of nodes in a layer using the fact that there at most w distinct functions among the restrictions of f.

We describe the i-th iteration of this construction. Let us say that the i-th layer of nodes has been constructed with at most w nodes. Say, the nodes are $\{v_{\boldsymbol{a}_1}, v_{\boldsymbol{a}_2}, \ldots, v_{\boldsymbol{a}_w}\}$ and they are supposed to compute the functions $\{f_{\boldsymbol{a}_1}, f_{\boldsymbol{a}_2}, \ldots, f_{\boldsymbol{a}_w}\}$ for some $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_w \in \{0,1\}^{i-1}$. For each $j \in [w]$, draw two edges from the node $v_{\boldsymbol{a}_j}$ labeled with 0 and 1 to two new nodes $v_{\boldsymbol{a}_j,0}$ and $v_{\boldsymbol{a}_j,1}$, respectively. The nodes $v_{\boldsymbol{a}_j,0}$ and $v_{\boldsymbol{a}_j,1}$ are supposed to compute the functions $f_{\boldsymbol{a}_j,0}$ and $f_{\boldsymbol{a}_j,1}$. We know that among these $2w$ functions $\{f_{\boldsymbol{a}_j,e} \mid j \in [w],\ e \in \{0,1\}\}$, at most w are distinct. One can simply merge two nodes $v_{\boldsymbol{a}_j,e}$ and $v_{\boldsymbol{a}_{j'},e'}$ if the functions $f_{\boldsymbol{a}_j,e}$ and $f_{\boldsymbol{a}_{j'},e'}$ are the same. Thus, we get at most w nodes in the $(i+1)$-th layer.

Repeating this procedure for n iterations gives us the ROBP. For any $\boldsymbol{a} \in \{0,1\}^n$, $f_{\boldsymbol{a}}$ can be either 0 or 1. Thus, we need only two nodes in the last layer. $\square$

Note that the construction of ROBP in the above lemma can be done for any function f, provided an oracle for testing $f_{\boldsymbol{a}} = f_{\boldsymbol{b}}$ for any $\boldsymbol{a}, \boldsymbol{b}$. Moreover, the construction requires at most $2w^2n$ accesses to this oracle. Rest of the procedure works in polynomial time.

From the construction in the above lemma, the following observation follows easily.

Observation 4.3 ([Bry86]). *For any function f, there is a unique minimal ROBP (when each layer has the minimum possible number of nodes) in a given variable order.*

In fact, the construction in Lemma 4.2 gives this minimal ROBP. This observation forms the backbone of the equivalence test for two ROBPs.

4.2 Equivalence of Two ROBPs

Savický and Wegener [SW97] considered the following question: given two ROBPs, with variables orders π' and π, computing the functions f and g, respectively, test whether $f = g$. Their idea is to find the minimal ROBPs for both f and g in the same variable order, say π. From Observation 4.3, the two ROBPs must be the same if f and g are equal. This can be tested easily. Note that when f and g are different, it is possible that the minimal ROBP for f in the variable order π has exponential size. In that case, we do not need to construct the ROBP completely. We can stop the ROBP construction for f as soon as we find that the number of nodes in some layer is larger than the width of the other ROBP computing g. If this happens then certainly $f \neq g$.

Now, the only thing left is the construction of the minimal ROBP. Recall from the previous section that this construction can be done for any function f, provided an oracle to test $f_{\boldsymbol{a}} = f_{\boldsymbol{b}}$ for some $\boldsymbol{a}, \boldsymbol{b} \in \{0,1\}^i$, for any $i \in [n]$. Given an ROBP for f in any variable order, one can easily construct an ROBP for $f_{\boldsymbol{a}}$.

Claim 4.4. *Given a width-w ROBP computing the function f, one can construct a width-w ROBP computing the function $f_{\boldsymbol{a}}$ in the same variable order (after removing the variables involved in the restriction $\boldsymbol{a}$), for any $\boldsymbol{a} \in \{0,1\}^i$ and any $i \in [n]$.*

To see this, consider the example when we have a restriction $x_j = 0$. In the layer corresponding to variable x_j, delete all the outgoing edges labeled with 1. As this layer now has only one outgoing edge for each node, one can delete this layer. To elaborate, let v be a node in the x_j layer and the 0-edge from v goes to the node v_0. We can simply delete the node v and direct all its incoming edges to v_0. This new ROBP will compute the restriction $f(x_j = 0)$. Doing this for all the restrictions given by $\boldsymbol{a}$ gives us an ROBP for $f_{\boldsymbol{a}}$. Clearly, the width does not increase in this process.

Note that the ROBPs for $f_{\boldsymbol{a}}$ and $f_{\boldsymbol{b}}$ would be in the same variable order, as $\boldsymbol{a}$ and $\boldsymbol{b}$ are restrictions on the same set of variables. Now, we come back to the question of testing $f_{\boldsymbol{a}} = f_{\boldsymbol{b}}$.

Claim 4.5. *Given two ROBPs in the same variable order, one can test whether they compute the same function in polynomial time.*

Bryant [Bry86] and Savický and Wegener [SW97] do this by minimizing the given ROBPs. However, we present a bit different approach here, which will also help us in the case of constantly many ROBPs. We rephrase the question as testing whether $f_{\boldsymbol{a}} \oplus f_{\boldsymbol{b}} = 0$. The next lemma shows that for the function $f_{\boldsymbol{a}} \oplus f_{\boldsymbol{b}}$, one can construct a small ROBP. Thus, our question will reduce to testing whether a given ROBP computes the zero function.

Lemma 4.6 ([Bry86]). *Let f and g be functions computed by two width-w ROBPs A and B, respectively, in the same variable order. Then we can construct a width-w^2 ROBP for $f \oplus g$ in the same variable order, in polynomial time.*

Proof. Let both the ROBPs have n variables. We will build an ROBP C which 'remembers' the states of both the ROBPs A and B. For each $1 \leq i \leq n+1$, the i-th layer of vertices in C is the cross product of vertices in the i-th layers of A and B. That is the set

$$\{(u, v) \mid u \text{ and } v \text{ are vertices in the } i\text{-th layer of } A \text{ and } B\text{, respectively}\}.$$

The start node of C will be (s_A, s_B), where s_A and s_B are the start nodes of A and B, respectively. There is an edge from (u, v) to (u', v') labeled with $e \in \{0, 1\}$ if and only if there is an edge from u to u' and an edge from v to v', both labeled with e. It is easy to see that for any $\boldsymbol{a} \in \{0, 1\}^i$, if the path corresponding to $\boldsymbol{a}$ ends in the node u in A and in the node v in B, then the corresponding path in C ends in (u, v).

Let the accepting and the rejecting nodes of A be denoted by t_1^A and t_0^A, respectively. Similarly, for B they are denoted by t_1^B and t_0^B. The last layer of C will consist of four nodes $\{(t_1^A, t_1^B), (t_0^A, t_1^B), (t_1^A, t_0^B), (t_0^A, t_0^B)\}$. It is easy to see that the path corresponding to any string $\boldsymbol{a} \in \{0,1\}^n$ in the ROBP C will end in

$$\begin{cases} (t_1^A, t_1^B) & \text{if } f(\boldsymbol{a}) = 1 \text{ and } g(\boldsymbol{a}) = 1, \\ (t_1^A, t_0^B) & \text{if } f(\boldsymbol{a}) = 1 \text{ and } g(\boldsymbol{a}) = 0, \\ (t_0^A, t_1^B) & \text{if } f(\boldsymbol{a}) = 0 \text{ and } g(\boldsymbol{a}) = 1, \\ (t_0^A, t_0^B) & \text{if } f(\boldsymbol{a}) = 0 \text{ and } g(\boldsymbol{a}) = 0 \end{cases}.$$

As we want to compute $f \oplus g$, we simply have to merge the nodes (t_1^A, t_0^B) and (t_0^A, t_1^B) and mark it the accepting node. Similarly, merge the nodes (t_1^A, t_1^B) and (t_0^A, t_0^B) and mark it the rejecting node. □

It is easy to see that the construction in the above lemma can be done for not just $f \oplus g$, but any Boolean operation on f and g. Once we have an ROBP for $f \oplus g$, we can simply check whether there is a path from its starting node to its accepting node. There is no such path if and only if $f \oplus g = 0$. This proves Claim 4.5 and in turn finishes the equivalence test of two ROBPs (in different variable orders).

Theorem 4.7 ([SW97]). *Given two ROBPs, possibly in different variable orders, we can decide if they compute the same function, in polynomial time.*

4.3 XOR of constantly many ROBPs

In this section, we extend the result of Savický and Wegener [SW97] to the XOR of constantly many ROBPs. That is, we give a polynomial time algorithm to test if an XOR of constantly many ROBPs computes the zero function. Let $A_1, A_2, \ldots, A_c$ be width-w ROBPs (possibly in different variable orders) computing the functions $f_1, f_2, \ldots, f_c$. The goal is to decide whether $f_1 \oplus f_2 \oplus \cdots \oplus f_c = 0$. To rephrase it whether $f_c = f_1 \oplus f_2 \oplus \cdots \oplus f_{c-1}$.

The approach is same as in the case of two ROBPs. Let $g := f_1 \oplus f_2 \oplus \cdots \oplus f_{c-1}$. Let the variable order of A_c be π. The idea is to build minimal ROBPs for f_c and g in the variable order π (by the construction in Lemma 4.2). If we detect that the width of the ROBP computing g is large (more than w), then we can output $f_c \neq g$. Otherwise, we find the minimal ROBP for g, which is small, and compare it with the minimal ROBP for f_c. They must be the same if $g = f_c$ (Observation 4.3).

The construction of Lemma 4.2 requires an oracle for testing $g_{\boldsymbol{a}} = g_{\boldsymbol{b}}$ for some $\boldsymbol{a}, \boldsymbol{b}$. It is easy to see that $g_{\boldsymbol{a}} = f_{1\boldsymbol{a}} \oplus f_{2\boldsymbol{a}} \oplus \cdots \oplus f_{c-1\boldsymbol{a}}$. So, the question becomes whether

$$f_{1\boldsymbol{a}} \oplus f_{2\boldsymbol{a}} \oplus \cdots \oplus f_{c-1\boldsymbol{a}} = f_{1\boldsymbol{b}} \oplus f_{2\boldsymbol{b}} \oplus \cdots \oplus f_{c-1\boldsymbol{b}}.$$

By rearranging this equation, we get

$$(f_{1\boldsymbol{a}} \oplus f_{1\boldsymbol{b}}) \oplus \cdots \oplus (f_{c-1\boldsymbol{a}} \oplus f_{c-1\boldsymbol{b}}) = 0.$$

For each $j \in [c-1]$, $f_{j\boldsymbol{a}}$ and $f_{j\boldsymbol{b}}$ have a width-w ROBP in the same variable order (from Claim 4.4). Thus, from Lemma 4.6, $f_{j\boldsymbol{a}} \oplus f_{j\boldsymbol{b}}$ has a width-w^2 ROBP. Let h_j denote the function $f_{j\boldsymbol{a}} \oplus f_{j\boldsymbol{b}}$ for each $j \in [c-1]$. Now, the question is to test whether

$$h_1 \oplus h_2 \oplus \cdots \oplus h_{c-1} = 0.$$

where each h_j is computed by a width-w^2 ROBP.

By the above procedure, we have reduced the question of c ROBPs with width w to the question of $c-1$ ROBPs with width w^2. Recall that in the construction of Lemma 4.2, we need $2nw^2$ accesses to the oracle testing whether $g_{\boldsymbol{a}} = g_{\boldsymbol{b}}$. Thus, we get the following recursive relation

$$T(n, w, c) = 2nw^2 \cdot T(n, w^2, c-1) + \mathsf{poly}(n, w),$$

where $T(n, w, c)$ is the time needed to test whether an XOR of c ROBPs, with n variables and width w, is the zero function. Solving this recursive relation,

$$T(n, w, c) = \mathsf{poly}(n^c, w^{2^c}).$$

Theorem 4.8. *Given c ROBPs, possibly in different variable orders, with n variables and width w, computing the functions $f_1, f_2, \ldots, f_c$, one can decide in time* $\mathsf{poly}(n^c, w^{2^c})$

whether $f_1 \oplus f_2 \oplus \cdots \oplus f_c = 0$, or in other words, whether $f_1 \oplus f_2 \oplus \cdots \oplus f_c$ is unsatisfiable.

4.4 PIT for Sum of constantly many ROABPs

ROABPs have a characterization similar to ROBPs, using partial evaluations. Instead of the number of distinct functions, here one has to find the number of linearly independent polynomials among the partial evaluations. Thus, we have an analogous result for sum of ROABPs over any field.

Theorem 4.9 ([GKST15])**.** *Given c ROABPs, possibly in different variable orders, with n variables, width w and individual degree d, computing the polynomials $f_1, f_2, \ldots, f_c \in \mathbb{F}[\boldsymbol{x}]$, one can decide whether $f_1 + f_2 + \cdots + f_c = 0$ in time $\mathsf{poly}(n^c, d^c, w^{2^c})$.*

The algorithm is exactly along the same lines as in the Boolean setting. Here, instead of testing the equality $g_{\boldsymbol{a}} = g_{\boldsymbol{b}}$, one needs to test linear dependencies of the type $\sum_j \alpha_j g_{\boldsymbol{a}_j} = 0$. In [GKST15], we also gave a blackbox PIT for the same class of polynomials, but with quasi-polynomial time complexity. Detailed proofs can be found in Korwar's thesis [Kor15].

4.5 Discussion

The first question is whether one can reduce the time complexity for XOR of c ROBPs from $w^{O(2^c)}$ to $w^{O(c)}$. This might also give some ideas for the PIT question. Note that when all the c ROBPs are in the same variable order, then one can test the zeroness in time $w^{O(c)}$ (from Lemma 4.6).

In the other direction, one can try to show that for some large enough c, satisfiability for XOR of c ROBPs (different variable orders) is NP-hard.

As mentioned before, the idea for equivalence of two ROABPs was inspired by the equivalence of two ROBPs. It would be interesting to know if there are concrete connections between arithmetic and Boolean branching programs. In particular, can ideas from identity testing of an ROABP be applied to construct pseudorandomness for ROBPs.

Chapter 5

Better Hitting-sets for Special Cases of ROABP

In this chapter, we give better hitting-sets for two special cases of ROABP. First is the case of an ROABP with known variable order. The best hitting-set known for this case had cost $(ndw)^{O(\log n)}$ [FS13b], where n is the number of variables, d is the individual degree and w is the width of the ROABP. Even for a constant-width ROABP, nothing better than a quasi-polynomial bound was known. We improve the hitting-set complexity for the known-order case to $\mathsf{poly}(ndw^{\log n})$, or equivalently, $\mathsf{poly}(dn^{\log w})$. In particular, this gives the first polynomial time hitting-set for constant-width ROABP (known-order). However, our hitting-set works only over those fields whose characteristic is zero or quasi-polynomially large.

The second case we consider is that of commutative ROABP. The best hitting-set known for this case had cost $d^{O(\log w)}(nw)^{O(\log\log w)}$ [FSS14]. We improve this hitting-set complexity to $(ndw)^{O(\log\log w)}$.

5.1 Hitting-set for Known-order ROABP

Our idea is inspired by the pseudorandom generator construction of Impagliazzo, Nisan and Wigderson [INW94] for read-once ordered Boolean branching programs (defined in

Chapter 4). First, we give an overview of their idea and describe how our result is analogous to theirs. The next subsection in not essential for the construction of the hitting-set and can be skipped.

5.1.1 Pseudorandom Generators for ROBP

A pseudorandom distribution for a class of functions is a distribution which cannot be distinguished from the uniform distribution by any function of that class. $\mathcal{G}\colon \{0,1\}^\ell \to \{0,1\}^n$ is a pseudorandom generator (PRG) with seed length ℓ, if $\mathcal{G}(U_\ell)$ is a pseudorandom distribution, where U_ℓ is the uniform distribution on $\{0,1\}^\ell$. Formally, $\mathcal{G}$ is a PRG for a class of Boolean functions $\mathcal{F}$, if for every function $f\colon \{0,1\}^n \to \{0,1\}$ from the class $\mathcal{F}$,

$$\left| \Pr_{x\in U_n}[f(x)=1] - \Pr_{y\in U_\ell}[f(\mathcal{G}(y))=1] \right| \leq 1/n^3.$$

The result of Impagliazzo, Nisan and Wigderson [INW94] implies an $O(\log^2 n)$ seed-length PRG for ROBPs (known variable order). The basic building block of their construction is as follows. Let us say the variable order of the ROBP is $(x_1, x_2, \ldots, x_n)$ and it computes the function $f(x_1, x_2, \ldots, x_n)$. They divide the n-variate ROBP in two halves reading the input bits $(x_1, \ldots, x_{n/2})$ and $(x_{n/2+1}, \ldots, x_n)$, respectively. Let s and t be the source and the sink (accepting) node of the ROBP, respectively and $v_1, v_2, \ldots, v_w$ be the nodes in the $(n/2+1)$-th layer (w is the width). Any assignment of the variables $x_1, x_2, \ldots, x_{n/2}$ gives us a path in the ROBP which leads to one of the nodes in $\{v_i\}_i$. Let the function computed from the node v_i to t be $h_i(x_{n/2+1}, \ldots, x_n)$ for each $i \in [w]$. After any assignment of the first $n/2$ variables, the function f becomes one of the functions in $\{h_i\}_{i=1}^w$. Thus, the function $f(x_1, x_2, \ldots, x_n)$ can be written as

$$f = \bigvee_{i=1}^{w} (g_i \wedge h_i), \tag{5.1}$$

where the function $g_i(x_1, \ldots, x_{n/2})$ is true exactly for those input assignments which lead to the node v_i in the ROBP for each $i \in [w]$. Impagliazzo, Nisan and Wigderson [INW94] gave a PRG with seed length $r/2 + O(\log w)$ for any function of the form given by Equation (5.1), when each g_i, h_i is an $r/2$-bit function. That is, they constructed functions

$\mathcal{G}_0, \mathcal{G}_1 \colon \{0,1\}^{r/2+\ell} \to \{0,1\}^{r/2}$, for $\ell = O(\log w)$, such that

$$\left| \Pr_{y_1,y_2 \in U_{r/2}} [f(y_1,y_2)=1] - \Pr_{y \in U_{r/2+\ell}} [f(\mathcal{G}_0(y), \mathcal{G}_1(y))=1] \right| \leq 1/w. \tag{5.2}$$

The constructed PRG has sample space of size only $2^{r/2+\ell} = 2^{r/2} \cdot \mathsf{poly}(w)$ as compared to the original sample space size 2^r. To get the final construction, [INW94] apply this procedure of replacing $(r/2, r/2)$ random bits with $r/2 + \ell$ random bits, recursively in a bottom-up fashion, starting from $r = O(\log n)$.

For the PIT question, we start with a polynomial of a form analogous to Equation (5.1).

$$f(x_1,x_2) = \sum_{i=1}^{w} g_i(x_1) h_i(x_2). \tag{5.3}$$

Here, we take the polynomials g_is and h_is as univariates, as a multivariate polynomial can be converted to a univariate polynomial with an appropriately high degree, while preserving the non-zeroness. If the degree of each g_i and h_i is bounded by d then there is a trivial hitting-set for $f(x_1,x_2)$ of size $(d+1)^2$ (Lemma 2.2). Here, our goal is to get a hitting-set of size $d \cdot \mathsf{poly}(w)$ (as in the Boolean setting).

To do this, we simply show that for any polynomial f of the form given by Equation (5.3), the polynomial $f(t^w, t^w + t^{w-1})$ is nonzero (for zero characteristic fields). This substitution gives a nonzero univariate polynomial of degree $2dw$, which has a hitting-set of size $2dw+1$. Applying this bivariate to univariate reduction recursively to an n-variate ROABP, in $\log n$ rounds, will give us a nonzero univariate polynomial with degree $ndw^{\log n}$, and thus, a hitting-set of the same size. For constant width ROABP, the complexity becomes $\mathsf{poly}(n,d)$.

Note that in the Boolean setting, the recursive procedure of [INW94] does not lead to a PRG with a polynomial size sample space (or $O(\log n)$ seed length), even when the width of the ROBP is constant. The reason is that the error probability in (5.2) is $1/w$ and to make it $1/\mathsf{poly}(n)$, one needs $r/2 + O(\log n)$ seed length instead of $r/2 + O(\log w)$. The error probability needs to be $1/\mathsf{poly}(n)$, because in the lower levels of the recursion, the PRG needs to work for many functions simultaneously. The total error probability is upper bounded by the union bound, and hence, the error probability in one step cannot be independent of n.

Forbes and Shpilka [FS13b] have applied a similar scheme for the arithmetic setting. They also have the problem of a union bound, when they want their hitting-set (or rank extractor) to work for many polynomials simultaneously. Thus, their final hitting-set complexity becomes $n^{O(\log n)}$, even for the constant-width case.

In contrast, our map $(x_1, x_2) \mapsto (t^w, t^w + t^{w-1})$ works for every polynomial of the form (5.3) simultaneously. Thus, there is no need for applying a union bound. Our technique has another crucial difference from previous works on ROABP ([FSS14, FS13b] and Chapter 3). The basic building block in all the previous techniques is a monomial map, i.e., each variable is mapped to a univariate monomial. On the other hand we use a polynomial map. It is possible that our ideas for the arithmetic setting can help constructing an optimal PRG for constant-width ROBP.

5.1.2 Bivariate ROABP

To construct a hitting-set for ROABPs, we start with the bivariate case. Recall that a bivariate ROABP is of the form $U^\top D_1(x_1) D_2(x_2) T$, where $U, T \in \mathbb{F}^{w \times 1}$, $D_1 \in \mathbb{F}^{w \times w}[x_1]$ and $D_2 \in \mathbb{F}^{w \times w}[x_2]$. It is easy to see that a bivariate polynomial $f(x_1, x_2)$ computed by a width-w ROABP can be written as $f(x_1, x_2) = \sum_{r=1}^{w} g_r(x_1) h_r(x_2)$. To give a hitting-set for this, we will use the notion of a partial derivative matrix defined by Nisan [Nis91] in the context of lower bounds. Let f have its individual degree bounded by d. The *partial derivative matrix* M_f for f is a $(d+1) \times (d+1)$ matrix with

$$M_f(i, j) = \text{coef}_f(x_1^i x_2^j),$$

for all $i, j \in [\![d]\!]$. It is well known that rank of M_f is equal to the smallest possible width of an ROABP computing f [Nis91].

Lemma 5.1 (rank $\leq$ width). *For any polynomial* $f(x_1, x_2) = \sum_{r=1}^{w} g_r(x_1) h_r(x_2)$,

$$\text{rank}(M_f) \leq w.$$

Proof. Let us define $f_r = g_r h_r$, for all $r \in [w]$. Clearly, $M_f = \sum_{r=1}^{w} M_{f_r}$, as $f = \sum_{r=1}^{w} f_r$.

We will show that $\mathrm{rank}(M_{f_r}) \leq 1$, for all $r \in [w]$. As $f_r = g_r(x_1)h_r(x_2)$, its coefficients can be written as a product of coefficients from g_r and h_r, i.e.,

$$\mathrm{coef}_{f_r}(x_1^i x_2^j) = \mathrm{coef}_{g_r}(x_1^i)\,\mathrm{coef}_{h_r}(x_2^j).$$

Now, it is easy to see that

$$M_{f_r} = u_r v_r^{\mathsf{T}},$$

where $u_r, v_r \in \mathbb{F}^{d+1}$ with $u_r = (\mathrm{coef}_{g_r}(x_1^i))_{i=0}^d$ and $v_r = (\mathrm{coef}_{h_r}(x_2^i))_{i=0}^d$.

Thus, $\mathrm{rank}(M_{f_r}) \leq 1$ and $\mathrm{rank}(M_f) \leq w$. $\square$

One can also show that if $\mathrm{rank}(M_f) = w$ then there exists a width-w ROABP computing f. We skip this proof as we will not need it. Now, using the above lemma we give a hitting-set for bivariate ROABPs.

Lemma 5.2. *Let* $\mathrm{char}(\mathbb{F}) = 0$*, or* $\mathrm{char}(\mathbb{F}) > d$*. Let* $f(x_1, x_2) = \sum_{r=1}^{w} g_r(x_1)h_r(x_2)$ *be a nonzero bivariate polynomial over* $\mathbb{F}$ *with individual degree* d*. Then* $f(t^w, t^w + t^{w-1}) \neq 0$.

Proof. Let $f'(t)$ be the polynomial after the substitution, i.e., $f(t^w, t^w + t^{w-1})$. Any monomial $x_1^i x_2^j$ will be mapped to the polynomial $t^{wi}(t^w + t^{w-1})^j$, under the mentioned substitution. The highest power of t coming from this polynomial is $t^{w(i+j)}$. We will cluster together all the monomials for which this highest power is the same, i.e., $i + j$ is the same. The coefficients corresponding to any such cluster of monomials will form a *diagonal* in M_f. The set $\{M_f(i,j) \mid i + j = k\}$ is defined to be the k-th diagonal of M_f, for all $0 \leq k \leq 2d$. Let ℓ be the highest number such that the ℓ-th diagonal has at least one nonzero element, i.e.,

$$\ell = \max\{i + j \mid M_f(i,j) \neq 0\}.$$

As $\mathrm{rank}(M_f) \leq w$ (from Lemma 5.1), we claim that the ℓ-th diagonal has at most w nonzero elements. To see this, let $\{(i_1, j_1), (i_2, j_2), \ldots, (i_{w'}, j_{w'})\}$ be the set of indices where the ℓ-th diagonal of M_f has nonzero elements, i.e., the set $\{(i,j) \mid M_f(i,j) \neq 0,\ i + j = \ell\}$. As $M_f(i,j) = 0$ for any $i + j > \ell$, it is easy to see that the rows $\{M_f(i_1), M_f(i_2), \ldots, M_f(i_{w'})\}$ are linearly independent. Thus, $w' \leq \mathrm{rank}(M_f) \leq w$.

Now, we claim that there exists an r with $w(\ell-1) < r \leq w\ell$ such that $\mathrm{coef}_{f'}(t^r) \neq 0$. To see this, first observe that the highest power of t which any monomial $x_1^i x_2^j$ with $i+j < \ell$ can contribute is $t^{w(\ell-1)}$. Thus, for any $w(\ell-1) < r \leq w\ell$, the term t^r can come only from the monomials $x_1^i x_2^j$ with $i+j \geq \ell$. We can ignore the monomials $x_1^i x_2^j$ with $i+j > \ell$ as $M_f(i,j) = 0$, when $i+j > \ell$. Now, for any $i+j = \ell$, the monomial $x_1^i x_2^j$ goes to

$$t^{w(\ell-j)}(t^w + t^{w-1})^j = \sum_{p=0}^{j} \binom{j}{p} t^{w\ell-p}.$$

Hence, for any $0 \leq p < w$,

$$\mathrm{coef}_{f'}(t^{w\ell-p}) = \sum_{a=1}^{w'} M_f(i_a, j_a)\binom{j_a}{p}.$$

Writing this in the matrix form we get

$$[\mathrm{coef}_{f'}(t^{w\ell}) \ \cdots \ \mathrm{coef}_{f'}(t^{w\ell-w+1})] = [M_f(i_1,j_1) \ \cdots \ M_f(i_{w'}, j_{w'})]C,$$

where C is a $w' \times w$ matrix with $C(a,b) = \binom{j_a}{b-1}$, for all $a \in [w']$ and $b \in [w]$. If all the rows of C are linearly independent then clearly, $\mathrm{coef}_{f'}(t^r) \neq 0$ for some $w(\ell-1) < r \leq w\ell$. We show the linear independence in Lemma 5.3. □

To finish the proof, we only need to show that the matrix C in the proof of Lemma 5.2 has all independent rows. To show this we need to assume that the numbers $\{j_a\}_a$ are all distinct. Hence, we need the field characteristic to be zero or strictly greater than d, as j_a can be as high as d for some $a \in [w']$.

Lemma 5.3. *Let C be a $w \times w$ matrix with $C(a,b) = \binom{j_a}{b-1}$, for all $a \in [w]$ and $b \in [w]$, where $\{j_a\}_a$ are all distinct numbers. Then C has full rank.*

Proof. We will show that for any nonzero $\alpha := (\alpha_1, \alpha_2 \ldots, \alpha_w) \in \mathbb{F}^{w\times 1}$, $C\alpha \neq 0$. Consider the polynomial $h(y) = \sum_{b=1}^{w} \alpha_b \frac{y(y-1)\cdots(y-b+2)}{(b-1)!}$. As $h(y)$ is nonzero polynomial with degree bounded by $w-1$, it can have at most $w-1$ roots. Thus, there exists an $a \in [w]$ such that $h(j_a) = \sum_{b=1}^{w} \alpha_b \binom{j_a}{b-1} \neq 0$. □

As mentioned above, the hitting-set proof works only when the field characteristic is zero or greater than d. We given an example over a small characteristic field, which demonstrates that the problem is not with the proof technique, but with the hitting-set itself. Let the field characteristic be 2. Consider the polynomial $f(x_1, x_2) = x_2^2 + x_1^2 + x_1$. Clearly, f has a width-2 ROABP. For a width-2 ROABP, the map in Lemma 5.2 would be $(x_1, x_2) \mapsto (t^2, t^2 + t)$. However, $f(t^2, t^2 + t) = 0$ (over $\mathbb{F}_2$). Hence, the hitting-set does not work.

Now, we move on to getting a hitting-set for an n-variate ROABP.

5.1.3 n-variate ROABP

Observe that the map given in Lemma 5.2 works irrespective of the degree of the polynomial, as long as the field characteristic is large enough. We plan to obtain a hitting-set for general n-variate ROABP by applying this map recursively. For this, we use the standard divide and conquer technique. First, we make pairs of consecutive variables in the ROABP. For each pair (x_{2i-1}, x_{2i}), we apply the map from Lemma 5.2, using a new variable t_i. Thus, we go to $n/2$ variables from n variables. In Lemma 5.4, we show that after this substitution the polynomial remains nonzero. Moreover, the new polynomial can be computed by a width-w ROABP. Thus, we can again use the same map on pairs of new variables. By repeating the halving procedure $\log n$ times we get a univariate polynomial. In each round the degree of the polynomial gets multiplied by w. Hence, after $\log n$ rounds, the degree of the univariate polynomial is bounded by $w^{\log n}$ times the original degree. Without loss of generality, let us assume that n is a power of 2. Recall that a width-w ROABP can be represented by the product $D_1 D_2 \cdots D_n$, where $D_1 \in \mathbb{F}^{1\times w}[x_1]$, $D_n \in \mathbb{F}^{w\times 1}[x_n]$ and $D_i \in \mathbb{F}^{w\times w}[x_i]$ for all $2 \le i \le n-1$.

Lemma 5.4 (Halving the number of variables)**.** *Let* $\mathrm{char}(\mathbb{F}) = 0$*, or* $\mathrm{char}(\mathbb{F}) > d$*. Let* $f(\boldsymbol{x}) = D_1(x_1)D_2(x_2)\cdots D_n(x_n)$ *be a nonzero polynomial computed by a width-*w *and individual degree-*d *ROABP, where* $D_1 \in \mathbb{F}^{1\times w}[x_1]$*,* $D_n \in \mathbb{F}^{w\times 1}[x_n]$ *and* $D_i \in \mathbb{F}^{w\times w}[x_i]$ *for all* $2 \le i \le n-1$*. Let the map* $\phi\colon \boldsymbol{x} \to \mathbb{F}[\boldsymbol{t}]$ *be such that for any index* $1 \le i \le n/2$*,*

$$\begin{aligned} \phi(x_{2i-1}) &= t_i^w, \\ \phi(x_{2i}) &= t_i^w + t_i^{w-1}. \end{aligned}$$

Then $f(\phi(\boldsymbol{x})) \neq 0$. Moreover, the polynomial $f'(t_1, t_2, \ldots, t_{n/2}) := f(\phi(\boldsymbol{x}))$ is computed by a width-w ROABP in the variable order $(t_1, t_2, \ldots, t_{n/2})$.

Proof. We will prove the lemma by induction on the number of variables.

Base Case (n=2): When there are only two variables, Lemma 5.2 proves the statement.

Induction Hypothesis: The statement is true for any $(n-2)$-variate ROABP.

Induction Step: We will prove the statement for the case of n variables. Let $G(x_1, x_2) := [g_1\ g_2\ \ldots\ g_w] = D_1 D_2$ and $H(x_3, x_4, \ldots, x_n) := [h_1\ h_2\ \ldots\ h_w]^\mathsf{T} = D_3 D_4 \cdots D_n$. Clearly, $f = GH$. We can assume that the polynomials $\{g_i\}_{i=1}^{w}$ are all linearly independent[1]. Because if it is not true then, as we argue below, one can modify the ROABP to have this property.

Without loss of generality, let $\{g_1, g_2, \ldots, g_{w'}\}$ be a maximal independent set for some $w' \leq w$. That is, for any $w' < j \leq w$, g_j is in the linear span of $\{g_i\}_{i=1}^{w'}$. Thus, there exists a matrix $\Gamma \in \mathbb{F}^{w' \times w}$ such that

$$G = [g_1\ g_2\ \ldots\ g_{w'}]\Gamma. \tag{5.4}$$

Consider the matrix $\Gamma_1 \in \mathbb{F}^{w \times w'}$ such that

$$\Gamma_1(i,j) = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to see that,

$$[g_1\ g_2\ \ldots\ g_{w'}] = G\Gamma_1. \tag{5.5}$$

From Equations (5.4) and (5.5), we can write

$$f = GH = G\Gamma_1\Gamma H.$$

Let us define $E_2 = D_2\Gamma_1$ and $E_3 = \Gamma D_3$. Clearly, the ROABP $D_1 E_2 E_3 D_4 \cdots D_n$ computes the polynomial f.

Now, we move on to prove that f remains nonzero under the map ϕ. Let us redefine $G(x_1, x_2) := [g_1\ g_2\ \ldots\ g_{w'}] = D_1 E_2$ and $H(x_3, x_4, \ldots, x_n) := [h_1\ h_2\ \ldots\ h_{w'}]^\mathsf{T} =$

[1]The polynomials $\{g_i\}_i$ are linearly independent if there is no nonzero $(\alpha_i)_i$ with $\sum_i \alpha_i g_i = 0$.

$E_3 D_4 \cdots D_n$. First we apply the map ϕ on the variables $\{x_i\}_{i=3}^{n}$. Observe that any h_i is a polynomial computed by an $(n-2)$-variate ROABP of width w, as $h_i = e_i^{\mathsf{T}} E_3 D_4 \cdots D_n$, where e_i is the i-th elementary unit vector. Hence, by induction hypothesis, any nonzero h_i would remain nonzero under the map ϕ. Thus,

$$H'(t_2, \ldots, t_{n/2}) := H(\phi(x_3), \phi(x_4), \ldots, \phi(x_n)) \neq [0\ 0\ \ldots\ 0]^{\mathsf{T}}.$$

This means there exists a monomial $\boldsymbol{t^a}$ in variables $\{t_i\}_{i=2}^{n/2}$ such that $\operatorname{coef}_{H'}(\boldsymbol{t^a}) \in \mathbb{F}^{w' \times 1}$ is a nonzero vector.

As the polynomials $\{g_i\}_{i=1}^{w'}$ are linearly independent,

$$G \operatorname{coef}_{H'}(\boldsymbol{t^a}) \neq 0.$$

Clearly, the polynomial $G \operatorname{coef}_{H'}(\boldsymbol{t^a}) = D_1 E_2 \operatorname{coef}_{H'}(\boldsymbol{t^a})$ is computed by a width-w bivariate ROABP in variables (x_1, x_2). Thus, it must remain nonzero under the map ϕ by Lemma 5.2. That is,

$$G'(t_1) \operatorname{coef}_{H'}(\boldsymbol{t^a}) = G(\phi(x_1), \phi(x_2)) \operatorname{coef}_{H'}(\boldsymbol{t^a}) \neq 0.$$

This implies that $f' = G'(t_1) H'(t_2, \ldots, t_{n/2}) \neq 0$. To see this, consider a monomial t_1^b such that

$$\operatorname{coef}_{G'}(t_1^b) \operatorname{coef}_{H'}(\boldsymbol{t^a}) \neq 0.$$

This product is nothing but $\operatorname{coef}_{f'}(t_1^b \boldsymbol{t^a})$.

Now, we argue that f' has a width w ROABP. Let $D_i' := D_{2i-1}(t_i^w) D_{2i}(t_i^w + t_i^{w-1})$ for all $1 \leq i \leq n/2$. Clearly, $D_1' D_2' \cdots D_{n/2}'$ is an ROABP computing f' in variable order $(t_1, t_2, \ldots, t_{n/2})$, as $D'_1 \in \mathbb{F}^{1 \times w}[t_1]$, $D'_{n/2} \in \mathbb{F}^{w \times 1}[t_{n/2}]$ and $D_i' \in \mathbb{F}^{w \times w}[t_i]$ for all $2 \leq i \leq n/2 - 1$. □

By applying the map ϕ in Lemma 5.4, we reduced an n-variate ROABP to an $(n/2)$-variate ROABP, while preserving the non-zeroness. The resulting ROABP has same width w, but the individual degree goes up to become $2dw$, where d is the original individual degree. As our map ϕ is degree independent, we can apply the same map again on the variables $\{t_i\}_{i=1}^{n/2}$. It is easy to see that when the map ϕ is repeatedly applied in this way $\log n$ times, we get a nonzero univariate polynomial of degree $ndw^{\log n}$. Next lemma puts it formally. For ease of notation, we use variable numbering from 0 to $n-1$. Let $p_0(t) = t^w$ and $p_1(t) = t^w + t^{w-1}$.

Lemma 5.5. *Let* $\mathrm{char}(\mathbb{F}) = 0$*, or* $\mathrm{char}(\mathbb{F}) > ndw^{\log n - 1}$*. Let* $f \in \mathbb{F}[\boldsymbol{x}]$ *be a nonzero polynomial, with individual degree* d*, computed by a width-*w *ROABP in variable order* $(x_0, x_1, \ldots, x_{n-1})$*. Let the map* $\phi \colon \{x_0, x_1, \ldots, x_{n-1}\} \to \mathbb{F}[t]$ *be such that for any index* $0 \leq i \leq n-1$*,*

$$\phi(x_i) = p_{i_1}(p_{i_2} \cdots (p_{i_{\log n}}(t))),$$

where $i_{\log n}\ i_{\log n - 1}\ \cdots\ i_1$ *is the binary representation of* i*.*

Then $f(\phi(\boldsymbol{x}))$ *is a nonzero univariate polynomial with degree* $ndw^{\log n}$*.*

Note that the map ϕ crucially uses the knowledge of the variable order. We need $\mathrm{char}(\mathbb{F}) > ndw^{\log n - 1}$ as in the last round when we are going from two variables to one, the individual degree is $ndw^{\log n - 1}$ and Lemma 5.2 requires $\mathrm{char}(\mathbb{F})$ to be higher than the individual degree. For a univariate polynomial, the standard hitting-set is to plug-in distinct field values as many as one more than the degree. Thus, we get

Theorem 5.6. *For an* n*-variate, individual degree* d *and width-*w *ROABP, there is a blackbox PIT with time complexity* $dn^{\log w + 1}$*, when the variable order is known and the field characteristic is zero or larger than* $ndw^{\log n - 1}$*.*

From this, we immediately get the following result for constant-width ROABPs. Note that when w is constant, the lower bound on the characteristic becomes $(\mathsf{poly}(n))$.

Corollary 5.7. *There is a polynomial time blackbox PIT for constant width ROABPs, with known variable order and field characteristic being zero (or polynomially large).*

5.2 Commutative ROABP

In this section, we give better hitting-sets for commutative ROABPs. Recall that an ROABP is commutative if the matrices involved in the matrix product come from a commutative algebra. To elaborate, a commutative ROABP is of the form $U^{\mathsf{T}} D_1 D_2 \cdots D_n T$, where $U, T \in \mathbb{F}^{w \times 1}$ and $D_i \in \mathbb{F}^{w \times w}[x_i]$ is a polynomial over a commutative subalgebra of $\mathbb{F}^{w \times w}$ for each i. In simple words, $D_i D_j = D_j D_i$ for any $i, j \in [n]$. As the order of variables does not matter for a commutative ROABP, we take the standard variable order $(x_1, x_2, \ldots, x_n)$. Like Chapter 3, here we work with the polynomial $D = D_1 D_2 \cdots D_n$ over the matrix algebra. With an abuse of notation, we say $D_1 D_2 \cdots D_n$ is an ROABP computing a polynomial over matrices.

Our approach is similar to that of Forbes, Saptharishi and Shpilka [FSS14], who gave a $d^{O(\log w)}(nw)^{O(\log \log w)}$-time hitting-set for width-w, n-variate commutative ROABPs with individual degree bound d. We improve the time complexity for this case to $(ndw)^{O(\log \log w)}$. Note that when both d and w are $O(n)$, the complexity of [FSS14] becomes $n^{O(\log n)}$ which is comparable to the hitting-set complexity for general ROABPs (Chapter 3). On the other hand, our hitting-set for the commutative case is significantly better than the general case.

Forbes, Saptharishi and Shpilka [FSS14] constructed the hitting-set in two steps. Their first step is to show that for a given width-w ROABP, $O(\log w)$-concentration can be achieved by a shift with cost $nd^{O(\log w)}$, which is polynomially bounded when d is small. Recall that a polynomial $D(\boldsymbol{x})$ over an algebra is said to be ℓ-concentrated if its coefficients of $(< \ell)$-support monomials span all its coefficients (Definition 3.2). And by a shift we mean replacing each variable x_i with $x_i + t_i$ for some $\{t_i\}_i$. The second step in [FSS14] is to show that if a given commutative ROABP is $O(\log w)$-concentrated then there is a hitting-set for it of size $(ndw)^{O(\log \log w)}$. We improve the first step by giving a shift with cost $(ndw)^{O(\log \log w)}$. Our improvement of the first step implies an $(ndw)^{O(\log \log w)}$-time hitting-set for commutative ROABP.

First, we elaborate the first step of Forbes, Saptharishi and Shpilka [FSS14]. To achieve concentration they use the idea of Agrawal, Saha and Saxena [ASS13], i.e., achieving concentration in small sub-circuits implies concentration in the whole circuit. The following lemma puts it formally.

Lemma 5.8 ([ASS13, FSS14])**.** *Let $D(\boldsymbol{x}) = D_1(x_1) D_2(x_2) \cdots D_n(x_n)$ be a product of univariate polynomials over a commutative algebra $\mathbb{A}_k$. Suppose there exists an ℓ such* *that for any $S \in [n]$ with $|S| = \ell$, the polynomial $\prod_{i\in S} D_i$ has ℓ-concentration. Then $D(\boldsymbol{x})$ has ℓ-concentration.*

Proof. For any set $S \subseteq [n]$, let us define a sub-circuit D_S of D as $\prod_{i\in S} D_i(x_i)$. We will show ℓ-concentration in all the sub-circuits D_S of D, using induction on the size of S.

Base Case: D_S is trivially ℓ-concentrated if $|S| < \ell$. In the case of $|S| = \ell$, D_S is ℓ-concentrated from the hypothesis in the lemma.

Induction Hypothesis: D_S has ℓ-concentration for any set S with $|S| < j$.

Induction Step: We will prove ℓ-concentration in D_S for a set S with $|S| = j$. Let $S = \{x_{i_1}, x_{i_2}, \ldots, x_{i_j}\}$. Consider a monomial $\boldsymbol{x}^{\boldsymbol{a}} = x_{i_1}^{a_1} x_{i_2}^{a_2} \cdots x_{i_j}^{a_j}$ with support from the set S. Without loss of generality let us assume $a_1 \neq 0$. Now, let the set $S' = S \setminus \{x_{i_1}\}$ and let the monomial $\boldsymbol{x}^{\boldsymbol{a}'} = \boldsymbol{x}^{\boldsymbol{a}}/x_{i_1}^{a_1}$. As $|S'| = j-1$, by the inductive hypothesis $D_{S'}$ is ℓ-concentrated. Thus,

$$\mathrm{coef}_{D_{S'}}(\boldsymbol{x}^{\boldsymbol{a}'}) \in \mathrm{span}\{\mathrm{coef}_{D_{S'}}(\boldsymbol{x}^{\boldsymbol{b}}) \mid \mathrm{Supp}(\boldsymbol{b}) \subseteq S',\ \mathrm{supp}(\boldsymbol{b}) < \ell\}. \tag{5.6}$$

It is easy to see that for any monomial $\boldsymbol{x}^{\boldsymbol{b}}$ with its support in S',

$$\mathrm{coef}_{D_S}(\boldsymbol{x}^{\boldsymbol{b}} x_{i_1}^{a_1}) = \mathrm{coef}_{D_{S'}}(\boldsymbol{x}^{\boldsymbol{b}})\, \mathrm{coef}_{D_{i_1}}(x_{i_1}^{a_1}).$$

Thus, by multiplying $\mathrm{coef}_{D_{i_1}}(x_{i_1}^{a_1})$ in (5.6), we get

$$\mathrm{coef}_{D_S}(\boldsymbol{x}^{\boldsymbol{a}}) \in \mathrm{span}\{\mathrm{coef}_{D_S}(\boldsymbol{x}^{\boldsymbol{b}} x_{i_1}^{a_1}) \mid \mathrm{Supp}(\boldsymbol{b}) \subseteq S',\ \mathrm{supp}(\boldsymbol{b}) < \ell\}.$$

Hence,

$$\mathrm{coef}_{D_S}(\boldsymbol{x}^{\boldsymbol{a}}) \in \mathrm{span}\{\mathrm{coef}_{D_S}(\boldsymbol{x}^{\boldsymbol{b}}) \mid \mathrm{Supp}(\boldsymbol{b}) \subseteq S,\ \mathrm{supp}(\boldsymbol{b}) \leq \ell\}. \tag{5.7}$$

Now, we claim that for any monomial $\boldsymbol{x}^{\boldsymbol{b}}$ with $\mathrm{Supp}(\boldsymbol{b}) \subseteq S$ and $\mathrm{supp}(\boldsymbol{b}) = \ell$,

$$\mathrm{coef}_{D_S}(\boldsymbol{x}^{\boldsymbol{b}}) \in \mathrm{span}\{\mathrm{coef}_{D_S}(\boldsymbol{x}^{\boldsymbol{c}}) \mid \mathrm{Supp}(\boldsymbol{c}) \subseteq S,\ \mathrm{supp}(\boldsymbol{c}) < \ell\}. \tag{5.8}$$

To see this, let T be the support of the monomial $\boldsymbol{x}^{\boldsymbol{b}}$. As $|T| = \ell$, D_T has ℓ-concentration. Thus,

$$\mathrm{coef}_{D_T}(\boldsymbol{x}^{\boldsymbol{b}}) \in \mathrm{span}\{\mathrm{coef}_{D_T}(\boldsymbol{x}^{\boldsymbol{c}}) \mid \mathrm{Supp}(\boldsymbol{c}) \subseteq T,\ \mathrm{supp}(\boldsymbol{c}) < \ell\}. \tag{5.9}$$

For any monomial $\boldsymbol{x}^{\boldsymbol{c}}$ with support in T, one can write

$$\mathrm{coef}_{D_S}(\boldsymbol{x}^{\boldsymbol{c}}) = \mathrm{coef}_{D_T}(\boldsymbol{x}^{\boldsymbol{c}}) \prod_{i \in S \setminus T} \mathrm{coef}_{D_i}(1).$$

Note that the commutativity of the underlying algebra is crucial for this. Thus, by multiplying $\left(\prod_{i \in S \setminus T} \mathrm{coef}_{D_i}(1)\right)$ in (5.9) we get (5.8).

By combining (5.8) with (5.7), we get

$$\mathrm{coef}_{D_S}(\boldsymbol{x}^{\boldsymbol{a}}) \in \mathrm{span}\{\mathrm{coef}_{D_S}(\boldsymbol{x}^{\boldsymbol{c}}) \mid \mathrm{Supp}(\boldsymbol{c}) \subseteq S,\ \mathrm{supp}(\boldsymbol{c}) < \ell\},$$

for any monomial $\boldsymbol{x}^{\boldsymbol{a}}$ with $\mathrm{Supp}(\boldsymbol{a}) \subseteq S$. This proves ℓ-concentration in D_S.

Taking $S = [n]$, we get ℓ-concentration in D. $\square$

Now, the goal is just to achieve ℓ-concentration in an ℓ-variate ROABP (computing a polynomial over the matrix algebra). We would remark here that for an ℓ-variate polynomial over a k-dimensional algebra, one can hope to achieve ℓ-concentration only when $\ell \geq \log(k+1)$. To see this, consider the polynomial $D(\boldsymbol{x}) = \prod_{i=1}^{\ell}(1 + v_i x_i)$ over a k-dimensional algebra such that $k > 2^\ell - 1$. Suppose the vector v_is are such that all the 2^ℓ coefficients of the polynomial D are linearly independent. There are only $2^\ell - 1$ coefficients of D with $(< \ell)$-support. Hence, they cannot span the whole coefficient space of D, whatever the shift we use.

[ASS13] and [FSS14] achieve ℓ-concentration in arbitrary ℓ-variate polynomials over a k-dimension algebra for $\ell = \log(k+1)$ by an appropriate shift of the variables. They do a shift by a univariate map $\phi : \boldsymbol{x} \to \mathbb{F}[t]$, i.e., they replace each x_i with $x_i + \phi(x_i)$. Their map ϕ is such that when it is naturally extended to all the monomials, i.e., $\phi(\prod_i x_i^{\gamma_i}) = \prod_i \phi(x_i)^{\gamma_i}$, all the monomials in the specified ℓ variables are mapped to distinct monomials (or linearly independent polynomials) under ϕ. The cost of such a map would be proportional to number of ℓ-variate monomials, which is $d^{O(\ell)} = d^{O(\log k)}$ (Lemma 2.3). A map which has this property for all size-ℓ subsets of n variables will have cost $nd^{O(\log k)}$. Note that after a shift by the univariate map, the new coefficients belong to $\mathbb{F}(t)^k$ and the concentration is over the field $\mathbb{F}(t)$.

We give a new shift with cost $(ndw)^{O(\log \ell)} = (ndw)^{O(\log \log w)}$, for a width-$w$, ℓ-variate ROABP (w^2 is the dimension of the underlying algebra). The cost has n as a parameter because the shift works for any size ℓ subset of n variables. Note that while the map of [ASS13, FSS14] works for an arbitrary ℓ-variate polynomial, our map works only for ℓ-variate ROABPs. The map we use is the basis isolating weight assignment for ROABPs from Chapter 3. We simply use the fact that for any polynomial over a k-dimensional algebra, shift by a basis isolating map achieves $\log(k+1)$-concentration. That is, we take the map $\phi(x_i) = t^{\mathrm{w}(x_i)}$, where w is a basis isolating weight assignment (Definition 3.3).

In [GKST15, Lemma 5.2], we showed that shifting by a basis isolating weight assignment achieves concentration (a proof can also be found in Korwar's thesis [Kor15]).

Lemma 5.9 (Isolation to concentration)**.** *Let $A(\boldsymbol{x})$ be a polynomial over a k-dimensional algebra $\mathbb{A}_k$. Let* w *be a basis isolating weight assignment for $A(\boldsymbol{x})$. Then $A(\boldsymbol{x} + t^{\mathrm{w}})$ is ℓ-concentrated, where $\ell = \lceil \log(k+1) \rceil$ and t^{w} denotes the n-tuple $(t^{\mathrm{w}(x_1)}, t^{\mathrm{w}(x_2)}, \ldots, t^{\mathrm{w}(x_n)})$.*

We now recall the construction of a basis isolating weight assignment for ROABP. Here, we present a slightly modified version of Lemma 3.6 which easily follows from it.

Lemma 5.10. *Let $\boldsymbol{x}$ be a set of n variables. Let $D(\boldsymbol{x}) = D_1(x_{i_1}) D_2(x_{i_2}) \cdots D_\ell(x_{i_\ell})$ be an ℓ-variate polynomial over a k-dimensional algebra $\mathbb{A}_k$. Then we can construct a basis isolating weight assignment for $D(\boldsymbol{x})$ with the cost being $(\mathsf{poly}(k,n,d))^{\log \ell}$, where d is the individual degree.*

The construction in the proof of Lemma 3.6 actually gives a family $\mathcal{B}$ of $(knd)^{O(\log \ell)}$ weight assignments such that for any ℓ-variate ROABP, at least one of them is basis isolating. However, we are interested in a single map which works for every ℓ-variate ROABP. As seen in Lemma 5.9, shift by the n-tuple t^{w} gives ℓ-concentration in $A(\boldsymbol{x})$ when w is basis isolating for $A(\boldsymbol{x})$. To get a single shift for every ROABP, we take a Lagrange Interpolation of all the n-tuples in the family $\{t^{\mathrm{w}}\}_{\mathrm{w} \in \mathcal{B}}$.

Let $\mathcal{F} = \{\boldsymbol{f}_1(t), \boldsymbol{f}_2(t), \ldots, \boldsymbol{f}_N(t)\}$ be this family of n-tuples, where $\boldsymbol{f}_i$ is given by $\{f_{i1}(t), f_{i2}(t), \ldots f_{in}(t)\}$ for each i. Here, $N = (knd)^{O(\log \ell)}$. Their degrees are bounded by

$D = \max\{\deg(f_{i,j}) \mid i \in [N] \text{ and } j \in [n]\} = (knd)^{O(\log \ell)}$. The family $\mathcal{F}$ can be generated in time $(knd)^{O(\log \ell)}$.

Let $\boldsymbol{L}(y,t) \in \mathbb{F}[y,t]^n$ be the Lagrange interpolation of $\mathcal{F}$. That is, for all $j \in [n]$,

$$L_j = \sum_{i \in [N]} f_{i,j}(t) \prod_{\substack{i' \in [N] \\ i' \neq i}} \frac{y - \alpha_{i'}}{\alpha_i - \alpha_{i'}},$$

where $\{\alpha_i\}_{i \in [N]}$ are distinct field elements. (Recall that we assume the field $\mathbb{F}$ to be large enough so that these elements exist.) Note that $L_j|_{y=\alpha_i} = f_{i,j}$. Thus, $\boldsymbol{L}|_{y=\alpha_i} = \boldsymbol{f}_i$. Also, $\deg_y(L_j) = N - 1$ and $\deg_t(L_j) \leq D$.

Lemma 5.11. *Let $A(\boldsymbol{x})$ be a polynomial over $\mathbb{A}_k$ and there exists an $\boldsymbol{f} \in \mathcal{F}$ such that $A'(\boldsymbol{x}, t) = A(\boldsymbol{x} + \boldsymbol{f}) \in \mathbb{A}_k(t)[\boldsymbol{x}]$ is ℓ-concentrated. Then, $A''(\boldsymbol{x}, y, t) = A(\boldsymbol{x} + \boldsymbol{L}) \in \mathbb{A}_k(y,t)[\boldsymbol{x}]$ is ℓ-concentrated.*

Proof. Let $\mathrm{rank}_{\mathbb{F}}\{\mathrm{coef}_A(\boldsymbol{x}^{\boldsymbol{a}}) \mid \boldsymbol{x}^{\boldsymbol{a}} \in M\} = k'$, for some $k' \leq k$, and $M_\ell = \{\boldsymbol{x}^{\boldsymbol{a}} \in M \mid \mathrm{supp}(\boldsymbol{a}) < \ell\}$. We need to show that $\mathrm{rank}_{\mathbb{F}(y,t)}\{\mathrm{coef}_{A''}(\boldsymbol{x}^{\boldsymbol{a}}) \mid \boldsymbol{x}^{\boldsymbol{a}} \in M_\ell\} = k'$.

Since $A'(\boldsymbol{x})$ is ℓ-concentrated, we have that $\mathrm{rank}_{\mathbb{F}(t)}\{\mathrm{coef}_{A'}(\boldsymbol{x}^{\boldsymbol{a}}) \mid \boldsymbol{x}^{\boldsymbol{a}} \in M_\ell\} = k'$. Recall that $A'(\boldsymbol{x})$ is an evaluation of A'' at $y = \alpha_i$, i.e., $A'(\boldsymbol{x}, t) = A''(\boldsymbol{x}, \alpha_i, t)$ for some α_i. Thus, for all $\boldsymbol{x}^{\boldsymbol{a}} \in M$, we have $\mathrm{coef}_{A'}(\boldsymbol{x}^{\boldsymbol{a}}) = \mathrm{coef}_{A''}(\boldsymbol{x}^{\boldsymbol{a}})|_{y=\alpha_i}$.

Let $C \in \mathbb{F}[t]^{k \times |M_\ell|}$ be the matrix whose columns are $\mathrm{coef}_{A'}(\boldsymbol{x}^{\boldsymbol{a}})$, for $\boldsymbol{x}^{\boldsymbol{a}} \in M_\ell$. Similarly, let $C' \in \mathbb{F}[y,t]^{k \times |M_\ell|}$ be the matrix whose columns are $\mathrm{coef}_{A''}(\boldsymbol{x}^{\boldsymbol{a}})$, for $\boldsymbol{x}^{\boldsymbol{a}} \in M_\ell$. Then we have $C = C'|_{y=\alpha_i}$.

As $\mathrm{rank}_{\mathbb{F}(t)}(C) = k'$, there is a $k' \times k'$ submatrix in C, say indexed by (R, T), such that $\det(C(R,T)) \neq 0$. Because $\det(C(R,T)) = \det(C'(R,T))|_{y=\alpha_i}$, it follows that $\det(C'(R,T)) \neq 0$. Hence, we have $\mathrm{rank}_{\mathbb{F}(y,t)}(C') = k'$. Thus, the $(< \ell)$-support coefficients of A'' span its coefficient space. $\square$

Using the Lagrange interpolation, we can construct a single shift which works for all ℓ-variate ROABPs.

Lemma 5.12. *Given n, d, w and $\ell = \log(w^2 + 1)$, in time $(ndw)^{O(\log \ell)}$ one can compute a polynomial tuple $\boldsymbol{f}(t) \in \mathbb{F}[t]^n$ of degree $(ndw)^{O(\log \ell)}$ such that for any ℓ-variate poly-*

nomial $A(\boldsymbol{x}) \in \mathbb{F}^{w\times w}[\boldsymbol{x}]$ of individual degree d that can be computed by an ROABP of width w, the polynomial $A(\boldsymbol{x}+\boldsymbol{f}(t))$ is ℓ-concentrated.

Proof. Note that the dimension k of the underlying algebra is bounded by w^2. After shifting the polynomial $A(\boldsymbol{x})$ of by $\boldsymbol{L}(y,t)$, its coefficients will be polynomials in y and t, with degree $d' = dn(ndw)^{O(\log \ell)}$. Consider the determinant polynomial $\det(C'(R,T))$ from Lemma 5.11. As $k' \leq k$, $\det(C'(R,T))$ has degree bounded by $d'' = kd'$.

Note that when we replace y by $t^{d''+1}$, it does not affect the non-zeroness of the determinant, and hence, the concentration is preserved. Thus, $\boldsymbol{f} = \boldsymbol{L}(t^{d''+1},t)$ is an n-tuple of univariate polynomials in t that fulfills the claim of the lemma. $\square$

Combining Lemma 5.8 and Lemma 5.12 we get the following.

Lemma 5.13. *Given n,d,w, one can compute an n-tuple $\boldsymbol{f}(t)$ with cost $(ndw)^{O(\log\log w)}$ such that for any n-variate, individual degree-d polynomial $D(\boldsymbol{x}) \in \mathbb{F}^{w\times w}[\boldsymbol{x}]$ computed by a width-w commutative ROABP, $D(\boldsymbol{x}+\boldsymbol{f}(t))$ is $O(\log w)$-concentrated.*

Note that if the polynomial $D(\boldsymbol{x}) \in \mathbb{F}^{w\times w}[\boldsymbol{x}]$ is ℓ-concentrated then the polynomial $C(\boldsymbol{x}) = U^{\mathsf{T}}DT$ is also ℓ-concentrated, where $U,T \in \mathbb{F}^{w\times 1}$. This is true because multiplying by U^{T} and T are linear operations. Recall that for polynomial $C(\boldsymbol{x}) \in \mathbb{F}[\boldsymbol{x}]$, $O(\log w)$-concentration means that there is a monomial with $O(\log w)$-support which has a nonzero coefficient.

Now, we move on to the second step of Forbes, Shpilka and Saptharishi [FSS14]. They give a $(ndw)^{O(\log\log w)}$-time hitting-set for an already $O(\log w)$-concentrated commutative ROABP. They do this by reducing the PIT question to an $O(\log w)$-variate ROABP [FSS14, Lemma 7.6].

Lemma 5.14 ([FSS14])**.** *Let $C(\boldsymbol{x}) \in \mathbb{F}[\boldsymbol{x}]$ be an n-variate, individual degree-d polynomial computed by a width-w ROABP. Suppose $C(\boldsymbol{x})$ has an $(\leq \ell)$-support monomial with a nonzero coefficient. Then, there is a $\mathsf{poly}(n,w,d)$-time computable m-variate map $\phi\colon \boldsymbol{x} \to \mathbb{F}[y_1,y_2,\ldots,y_m]$ such that $C(\phi(\boldsymbol{x}))$ is a nonzero polynomial with degree $< d^2n^4$, where $m = O(\ell^2)$. Moreover, $C(\phi(\boldsymbol{x}))$ is computed by a width-w, m-variate commutative ROABP.*

We know from Theorem 3.7 that an m-variate, width-w ROABP has an $(mdw)^{O(\log m)}$-time hitting-set. Combining Lemma 5.13 and Lemma 5.14 with this fact and putting $m = O(\log^2 w)$, we get the following.

Theorem 5.15. *There is an $(ndw)^{O(\log \log w)}$-time hitting-set for n-variate commutative ROABPs with width w and individual degree d.*

5.3 Discussion

For our first result (Theorem 5.6), there are three directions of improvement. Ideally, one would like to have all three at once.

1. Find a similar hitting-set for the unknown-order case. In fact, we conjecture that the same hitting-set (Lemma 5.5) works for the unknown-order case as well.
2. Get a hitting-set for all characteristic fields. It is easy to construct examples over small characteristic fields where our hitting-set does not work.
3. Reduce the time complexity to polynomial time. To achieve this, it seems one has to do away with the divide and conquer approach.

As mentioned earlier, the ideas here can help in finding a better PRG for ROBPs. In particular, it is a big open question to find an $O(\log n)$-seed-length PRG for constant-width ROBPs (analogous to Corollary 5.7). Note that in the context of PRGs, usually the variable order is assumed to be known.

Following our second result (Theorem 5.15), it would be interesting to achieve the same time complexity for set-multilinear circuits. Recall from Section 2.5 that set-multilinear circuits are subsumed by commutative sparse-factor ROABPs, but not by commutative ROABPs. Following the approach for commutative ROABP (Lemma 5.13), one can achieve $O(\log k)$-concentration in a set-multilinear circuit in time $n^{O(\log \log k)}$. However, it is not clear whether the second step of the hitting-set construction can be done for set-multilinear circuits, i.e., finding a better hitting-set by assuming that the polynomial is already concentrated (Lemma 5.14).

Chapter 6

Parallel Complexity of Bipartite Matching

In this chapter, we give an introduction to the matching problem and its parallel complexity, and present some preliminary ideas towards finding an efficient parallel (NC) algorithm for bipartite matching.

6.1 Preliminaries

6.1.1 Complexity Classes

The complexity class NC is used to study the inherent parallel complexity of a problem.

Definition 6.1. A problem is in the class NC if it can be solved in polylogarithmic time using polynomially many processors. In particular, it is said to be in NC^i if the time required is $O(\log^i n)$.

As one is interested in the theoretical limits of parallelization, it is assumed that the processors have a shared memory and can access any bit of the memory in constant time. An equivalent definition of the class NC^i is given by the set of all problems which have log-space uniform Boolean circuits with depth $\log^i n$ and size $n^{O(1)}$ (see [Pap94, Chapter 15] for details).

All the basic arithmetic operations and matrix operations are in NC. In particular, determinant computation is in NC^2 [Ber84, MV97]. Determinant computation plays a crucial role in the RNC algorithm of Mulmuley, Vazirani and Vazirani [MVV87] for matching. RNC stands for randomized NC, i.e., an NC algorithm which uses random bits. Another class related to determinant computation is the class SPL, which is a subset of NC^2.

Definition 6.2. A language L is in the class SPL if there is a non-deterministic log-space turing machine M such that

$$\#acc(M(x)) - \#rej(M(x)) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L, \end{cases}$$

where $\#acc(M(x))$ and $\#rej(M(x))$ are the number of accepting and rejecting paths of M on input x, respectively.

For an integer matrix whose determinant is promised to be 0 or 1, deciding whether it is 0 or 1 is in SPL. See [ARZ99] for more details on the class SPL.

6.1.2 Graphs and Matchings

$G(V, E)$ denotes a graph with the vertex set V and the edge set E. We start by defining matchings in a graph.

Definition 6.3 (Matching)**.** In a graph $G(V, E)$, a matching $M \subseteq E$ is a subset of edges with no two edges sharing an endpoint. A matching which covers every vertex is called a perfect matching.

The perfect matching problem (Decision-PM) asks whether a given graph has a perfect matching. In other words, say for a group of people we are given connections of the form 'person x and person y know each other' and the question is whether one can make pairs of persons known to each other such that every person is paired with someone. In the context of matchings, a natural class of graphs which has been widely studied is bipartite graphs. In this thesis, we will only deal with matchings in bipartite graphs.

Definition 6.4 (Bipartite Graphs)**.** A graph is bipartite if there exist two partitions of the vertices such that any edges connects a vertex from one partition to a vertex from the other partition.

Planar graphs is another special class of graphs for which the matching problem has been extensively studied.

Definition 6.5 (Planar Graphs)**.** A planar graph is a graph which can be drawn in a plane without its edges intersecting.

Various versions of the matching problem are known to have better solutions in case of planar graphs as compared to general graphs, especially, in terms of their parallel complexity. For example, counting the number of perfect matchings in planar graphs is in NC [Kas67, Vaz89], while it is #P-complete for general graphs [Val79]. Constructing a perfect matching in a bipartite planar graph is in NC [MN95, MV00, DKR10], while no such result is known for general graphs.

6.1.3 Matching Polytope

Matchings are also one of the well-studied objects in polyhedral combinatorics. Matchings have an associated polytope, called the perfect matching polytope. The perfect matching polytope is also important from the computational complexity perspective. In fact, it forms the basis of one of the NC algorithms for bipartite planar matching [MV00].

The perfect matching polytope $PM(G)$ of a graph $G(V, E)$ is a polytope in the edge space, i.e., $PM(G) \subseteq \mathbb{R}^{|E|}$. For any perfect matching M, consider its incidence vector $x_M \in \mathbb{R}^E$ given by

$$x_{Me} = \begin{cases} 1 & \text{if } e \in M, \\ 0 & \text{otherwise.} \end{cases}$$

This vector is referred to as a perfect matching point for any perfect matching M. The perfect matching polytope of a graph G is defined to be the convex hull of all the perfect matching points.

Definition 6.6. $PM(G) = \text{conv}\{x_M \mid M \text{ is a perfect matching in } G\}$.

It is well known that for a bipartite graph G, its perfect matching polytope has a simple description (see [LP86]).

Lemma 6.7. *For a bipartite graph G, a point $x \in \mathbb{R}^E$ is in $PM(G)$ if and only if*

$$\begin{aligned} \sum_{e \in \delta(v)} x_e &= 1 \; \forall v \in V, && (6.1) \\ x_e &\geq 0 \; \forall e \in E, && (6.2) \end{aligned}$$

where $\delta(v)$ denotes the set of edges incident on the vertex v.

It is easy to see that any perfect matching point will satisfy these two condition. In fact, all perfect matching points are vertices of this polytope. The non-trivial part is to show that any point satisfying these two conditions is in the perfect matching polytope [LP86, Chapter 7]. For general graphs, the polytope described by (6.1) and (6.2) can have vertices which are not perfect matchings. Thus, the description does not capture the perfect matching polytope for general graphs.

We hope to leverage this simple description of the bipartite matching polytope to find an NC algorithm for bipartite matching.

6.2 RNC algorithm for Search-PM [MVV87]

Let us first recall the RNC algorithm of Mulmuley, Vazirani and Vazirani [MVV87] for construction of a perfect matching (Search-PM). Though the algorithm works for any graph, we will only consider bipartite graphs here.

For any weight assignment $w\colon E \to \mathbb{Z}$ on the edges of a graph, the weight of a matching M is defined to be the sum of the weights of all the edges in M, i.e., $w(M) = \sum_{e \in M} w(e)$. Let G be a bipartite graph with vertex partitions $U = \{u_i\}_{i=1}^n$ and $V = \{v_i\}_{i=1}^n$, each with n nodes. Consider the following $n \times n$ matrix whose rows and columns are indexed by the vertices in U and V, respectively.

$$B(i,j) = \begin{cases} 2^{w(e)} \text{ if } e = (u_i, v_j) \in E, \\ 0 \text{ otherwise.} \end{cases},$$

where w is a weight assignment on the edges. The algorithm in [MVV87] computes the determinant of this matrix. By the definition of the determinant,

$$\det(B) = \sum_{\pi \in S_n} \operatorname{sgn}(\pi) \prod_{i=1}^{n} B(i, \pi(i)), \tag{6.3}$$

where S_n is the set of all permutations on $[n]$. Now, observe that each permutation π potentially has a corresponding perfect matching $M_\pi = \{(u_1, v_{\pi(1)}), (u_2, v_{\pi(2)}), \ldots, (u_n, v_{\pi(n)})\}$. M_π is a valid perfect matching only if all the mentioned edges are present in the graph. If for some j, the edge $(u_j, v_{\pi(j)})$ is not present in the graph then clearly, $\prod_{i=1}^{n} B(i, \pi(i)) = 0$. Thus, one can write the sum in Equation 6.3 as a sum over all perfect matchings.

$$\det(B) = \sum_{M \text{is a perfect matching in } G} \operatorname{sgn}(M) 2^{w(M)},$$

where $\operatorname{sgn}(M)$ is the sign of the corresponding permutation.

If the graph G does not have a perfect matching, then clearly, $\det(B) = 0$. However, even when the graph has perfect matchings, there can be cancellations due to $\operatorname{sgn}(M)$ and $\det(B)$ can become zero. To avoid such cancellations, one needs to design the weight function w cleverly. In particular, if w is such that the minimum weight perfect matching in the graph G is unique then $\det(B) \neq 0$. This is because the term $2^{w(M)}$ corresponding to the minimum weight perfect matching cannot be cancelled with other terms which are strictly higher powers of 2. Such a weight function is called an isolating weight assignment.

Definition 6.8 ([MVV87])**.** For a graph $G(V, E)$, a weight assignment $\mathrm{w}\colon E \to \mathbb{Z}$ is isolating if the minimum weight perfect matching (if one exists) in G is unique.

Given an isolating weight assignment for G, one can easily construct the minimum weight perfect matching in NC. Let M^* be the minimum weight perfect matching in G. For all edge $e \in E$, do the following: delete e from G and compute $\det(B)$. If the term $2^{w(M^*)}$ disappears from $\det(B)$, then $e \in M^*$. Doing this in parallel for each edge, we can find all the edges in M^*. Note that to compute the determinant efficiently, the entries in matrices should have $\mathsf{poly}(n)$ bits and thus, the weights on the edges should be polynomially bounded. As the determinant computation is in NC^2, this construction is also in NC^2.

To get an isolating weight assignment, Mulmuley, Vazirani and Vazirani [MVV87] simply assign each edge a random weight from a small range. Their isolation lemma says that with a good probability, a random weight assignment is isolating.

Lemma 6.9 (Isolation lemma [MVV87])**.** *Let G be a graph with n vertices. For each edge e, let its weight $w(e)$ be chosen randomly, independently from $[2n^2]$. Then*

$$\Pr[\textit{the minimum weight perfect matching in } G \textit{ is unique}] \geq 1/2.$$

Clearly, if one can get a deterministic NC construction of an isolating weight assignment, then Search-PM would fall in NC.

6.3 An Approach towards Matching in Bipartite Graphs

Despite much effort, no construction of an isolating weight assignment is known for general graphs or bipartite graphs, not even with sub-exponential weights. Our idea is to relax the condition of a unique minimum weight perfect matching. Instead, we ask if one can assign weights in a graph G such that the number of minimum weight perfect matchings is significantly smaller than the total number of perfect matchings. If this can be done, then we consider the new graph G' whose edge set is the union of all minimum weight perfect matchings in G. One can ask: is G' significantly smaller than G, in terms of number of edges? The answer turns out to be no. In fact, it is possible that G' is same as G, even when not all perfect matchings in G have the minimum weight. Figure 6.1 gives such an example. Interestingly, such graphs are always non-bipartite. For bipartite graphs we can show that G' does become strictly smaller than G. Theorem 6.10 shows precisely this.

Ideally, we would want that the number of edges in G' is only a constant fraction of that in G. If this can be done, then we can repeat this weight construction process for the new graph G'. After $O(\log n)$ iterations, we will be left with a graph with only one perfect matching. For now, we do not know a weight construction which will reduce the number of edges by a constant fraction. Theorem 6.10 just shows that the number of

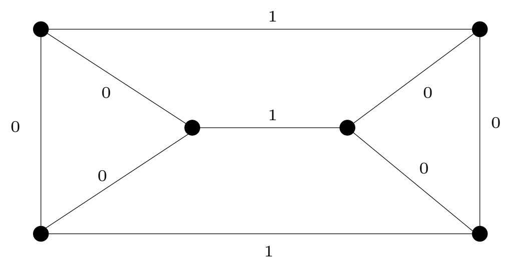

Figure 6.1: A graph with weighted edges. The minimum weight perfect matchings have weight 1 and their union contains all the edges of the graph.

edges definitely decreases, if not all perfect matchings are of minimum weight.

Theorem 6.10. *Let $G(V, E)$ be a bipartite graph with a weight function $w\colon E \to \mathbb{R}$ on its edges. Let E' be the union of all minimum weight perfect matchings in G according to w. Then every perfect matching in graph $G' = (V, E')$ has the same weight – the minimum weight of any perfect matching in G.*

Proof. For the proof we use the description of the perfect matching polytope for bipartite graphs (Lemma 6.7). A point $x \in \mathbb{R}^E$ is in $PM(G)$ if and only if

$$\sum_{e\in\delta(v)} x_e = 1 \;\forall v \in V, \tag{6.4}$$

$$x_e \geq 0 \;\forall e \in E, \tag{6.5}$$

Let us view w as a linear function on $\mathbb{R}^E$ by extending it in the natural way, i.e., for any $x \in \mathbb{R}^E$,

$$w(x) = \sum_{e\in E} x_e w(e).$$

Recall that x_M denotes the incidence vector of matching M. Clearly, for a perfect matching M, $w(M) = w(x_M)$. Let q be the weight of any minimum weight perfect matching in G. As the vertices of the matching polytope are all perfect matching points,

$$\min\{w(x) \mid x \in PM(G)\} = q.$$

In any polytope, the set of points minimizing a linear function form a face of the polytope. Let F be the face of the polytope $PM(G)$ consisting of all the points x with

$w(x) = q$.

One can describe a face of a polytope by replacing some of the inequalities in the description of the polytope by equalities. For the perfect matching polytope, the inequalities are given by (6.5). Thus, for the face F there must exist a set $S \subseteq E$ such that a point $x \in F$ if and only if

$$\begin{aligned} \sum_{e \in \delta(v)} x_e &= 1 \; \forall v \in V, && (6.6) \\ x_e &\geq 0 \; \forall e \in E \setminus S, && (6.7) \\ x_e &= 0 \; \forall e \in S. && (6.8) \end{aligned}$$

Clearly, for any minimum weight perfect matching M, the matching point x_M should satisfy the above three conditions as it lies on the face F. In particular, Equation (6.8) implies that for any $e \in S$, $e \notin M$. Hence, one can write,

$$E' \cap S = \emptyset,$$

where, E' is the union of all minimum weight perfect matchings.

Now, consider any perfect matching M' in the graph $G' = (V, E')$. It does not have any edges from the set S, hence $x_{M'}$ satisfies Equation (6.8). As M' is a perfect matching, $x_{M'}$ also satisfies (6.6) and (6.7). Thus, $x_{M'} \in F$. Hence, $w(M') = w(x_{M'}) = q$. This proves the theorem statement. □

Theorem 6.10 shows that in the union of all minimum weight perfect matchings, every perfect matching is of minimum weight. That is, taking the union does not create extra perfect matchings. Note that such a property would not hold for most combinatorial objects. For example, if one takes the union of all minimum weight spanning trees, the resulting graph will have other spanning trees which are not of the minimum weight. We hope that this special property of bipartite matching will help in finding an NC algorithm.

6.4 Discussion

The main question here is to find a weight assignment which ensures that the union of minimum weight perfect matchings has only a constant fraction of edges from the actual graph.

Chapter 7

Derandomizing Isolation Lemma for $K_{3,3}$-free or K_5-free Bipartite Graphs

As discussed in the previous chapter, an NC construction of an isolating weight assignment would put matching in NC. In this chapter, we give such a construction for two special classes of graphs, namely $K_{3,3}$-free bipartite graphs and K_5-free bipartite graphs.

Using the isolation lemma, Allender, Reinhardt and Zhou [ARZ99] showed that the DECISION-PM is in non-uniform SPL (SPL is defined in Section 6.1.1). The non-uniformity part is just for finding an isolating weight assignment. Hence, if an isolating weight assignment can be constructed in L, then DECISION-PM would be in SPL. Moreover, it would put SEARCH-PM in $\mathsf{FL}^{\mathsf{SPL}}$ (see [DKR10]). $\mathsf{FL}^{\mathsf{SPL}}$ is the set of function problems which can be solved by a log-space Turing machine with access to an SPL oracle.

Theorem 7.1. *Given a $K_{3,3}$-free or K_5-free bipartite graph, an isolating weight assignment (polynomially bounded) for it can be constructed in log-space.*

This theorem together with the results of [ARZ99] and [DKR10] gives us the following results about matching.

Corollary 7.2. *For a $K_{3,3}$-free or K_5-free bipartite graph,*

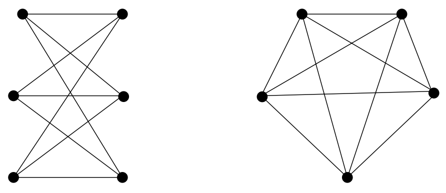
(a) $K_{3,3}$ (b) K_5

Figure 7.1: The graphs $K_{3,3}$ and K_5

– DECISION-PM *is in* SPL.

– SEARCH-PM *is in* $\mathsf{FL}^{\mathsf{SPL}}$.

– MIN-WEIGHT-PM *is in* $\mathsf{FL}^{\mathsf{SPL}}$.

The problem MIN-WEIGHT-PM asks to construct the minimum weight perfect matching in a given graph with polynomially bounded weights on its edges. Actually our results work for a general class of graphs defined via clique-sum operation, which subsumes $K_{3,3}$-free or K_5-free bipartite graphs.

7.1 Preliminaries

Our starting point is the result of Datta, Kulkarni and Roy [DKR10], which showed the same result for bipartite planar graphs. Let us recall the definition of a planar graph.

Definition 7.3 (Planar Graphs)**.** A planar graph is a graph which can be drawn on a plane without its edges intersecting.

Wagner [Wag37] gave an interesting characterization of planar graphs in terms of graph minors. A graph H is a minor of a graph G, if H can be obtained from G by deleting some vertices and edges and contracting some edges. Contraction of an edge means deleting the edge and identifying its two endpoints. G is called H-free if H is not a minor of G.

Theorem 7.4 ([Wag37])**.** *A graph is planar if and only if it is both $K_{3,3}$-free and K_5-free.*

$K_{3,3}$ is the complete bipartite graph on $(3,3)$ nodes and K_5 is the complete graph on 5 nodes (Figure 7.1). These two graphs are non-planar. Hence, if any graph G has either

$K_{3,3}$ or K_5 as its minor, then G would be non-planar. Wagner [Wag37] essentially showed that any non-planar graph must have a $K_{3,3}$ or K_5 as a minor. A natural generalization of planar graphs would be graphs which have one of these two graphs as a minor, but not the other one. Such a graph would be a $K_{3,3}$-free graph or a K_5-free graph.

Wagner [Wag37] and Asano [Asa85] gave exact characterizations of $K_{3,3}$-free graphs and K_5-free graphs, respectively, in terms of graph decomposition. They essentially showed that these graphs can be formed as combinations of planar graphs and constant-sized graphs, glued together by an operation called clique-sum (defined in Section 7.4). Actually, the class of graphs obtained by such combinations is bigger than $K_{3,3}$-free graphs and K_5-free graphs. Our techniques will also work for this bigger class.

7.1.1 Log-space Operations on Graphs

As we want our construction to be in log-space. We will need to do several standard graph operations in log-space. We describe these operations with appropriate references.

Reachability in an undirected graph, i.e., given two vertices u and v, deciding whether u and v are connected by a path, is in log-space due to the famous result of Reingold [Rei08]. Thus, finding connected components of a graph is also in log-space.

Allender and Mahajan [AM04] have shown that deciding whether a graph is planar and computing a planar embedding can be done in log-space with an oracle access to reachability. Together with Reingold's result [Rei08], it follows that both the things can be done in log-space.

For a given tree, we describe its log-space traversal which will be used to make the tree rooted. A tree is rooted when a root is fixed and every connected pair of vertices has a child-parent relationship with the one closer to the root being the parent.

Log-space tree traversal [Lin92]: Fix an arbitrary root r for the tree. For every node in the tree, give its edges an arbitrary cyclic ordering, i.e., every edge has a left and a right neighbor. Start traversing from the root r by taking an arbitrary edge. If you arrive at a node u using its edge e, then leave node u using the right neighbor of e. If e is the only edge on u, then leave using the same edge. Stop when you see the edge from where the travel started. It is easy to see that this traversal ends at the root node r with every edge being traversed exactly twice (in different directions). To find the parent of a node u, one just need to stop the traversal when the node u is reached for the first time. The previous node in the traversal will be the parent of u. Finding the parent for every node will give us the complete rooted tree.

Any node u and all its descendants form a subtree rooted at u. It is easy to count the number of nodes in the subtree rooted at u: Go over each node v and decide if it is a descendant of u, i.e., it appears after u in the tree traversal. Similarly, given a weight for every node one can compute the sum of weights of all the descendants of u. We will need to compute these functions on a given tree in our constructions.

7.1.2 Biconnected Graphs

If a graph G is disconnected then a perfect matching in G can be constructed by taking a union of perfect matchings in its different connected components. As connected components of a graph can be found in log-space [Rei08], we will always assume that the given graph is connected.

Let G be a connected graph. A vertex a in G is called an *articulation point*, if its removal will make G disconnected. A graph without any articulation point is called biconnected. Let a be an articulation point in G such that its deletion creates connected components $G_1, G_2, \dots, G_m$. It is easy to see that for G to have a perfect matching, exactly one of these components should have an odd number of vertices, say G_1. Then, in any perfect matching of G, the vertex a will always be matched to a vertex in G_1. Thus, we can delete any edge connecting a to other components, and all the perfect matchings will still be preserved. The above reduction can be done in log-space via reachability queries [Rei08, TW14] as follows: go over each vertex a and check whether its deletion disconnects the graph. If yes, then find out the cardinalities of different components after deletion of a. Now, delete any edge incident on a which connects it to a set of even size. Thus, we will always assume that the given graph is biconnected.

7.2 Isolation in Bipartite Planar Graphs [DKR10]

We first describe the approach of Datta, Kulkarni and Roy [DKR10] for bipartite planar graphs. Our presentation slightly differs from theirs and follows that of Korwar [Kor09].

Let us first define a skew-symmetric weight function on the edges of a graph. For this, we consider the edges of the graph directed in both the directions. We call this directed set of edges $\vec{E}$. A weight function $w\colon \vec{E} \to \mathbb{Z}$ is called skew-symmetric if for any edge (u, v), $w(u, v) = -w(v, u)$.

Definition 7.5 (Circulation)**.** For a cycle C, whose edges are given by $\{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k), (v_k, v_1)\}$, its circulation is defined to be $w(v_1, v_2)+w(v_2, v_3)+\cdots+w(v_k, v_1)$.

Clearly, as our weight function is skew-symmetric, changing the orientation of the cycle, only changes the sign of the circulation. The following lemma [TV12, Theorem 6] gives the connection between nonzero circulations and isolation of a matching (it appears in [DKR10] in a slightly different form). For a bipartite (undirected) graph $G(V_1, V_2, E)$, a skew-symmetric weight function $w\colon \vec{E} \to \mathbb{Z}$ on its edges, has a natural interpretation on the undirected edges as $\mathrm{w}\colon E \to \mathbb{Z}$ such that $\mathrm{w}(u, v) = w(u, v)$, where $u \in V_1$ and $v \in V_2$.

Lemma 7.6 ([DKR10, TV12])**.** *Let $w\colon \vec{E} \to \mathbb{Z}$ be a skew-symmetric weight function on the edges of a bipartite graph G such that every cycle has a non-zero circulation. Then, $\mathrm{w}\colon E \to \mathbb{Z}$ is an isolating weight assignment for G.*

The bipartiteness assumption is needed only in the above lemma. We will show a log-space construction of a skew-symmetric weight function that guarantees nonzero circulation for every cycle in a given planar graph, i.e., without assuming bipartiteness.

A planar embedding of a given planar graph can be computed in log-space [AM04, Rei08]. We start with a planar embedding of the graph. For any weight assignment $w : \vec{E} \to \mathbb{Z}$ on the edges of the graph, we define the *circulation of a face* as the circulation of the corresponding cycle in the clockwise direction. More formally,

Definition 7.7 (Circulation of a face)**.** Let $G(V, E)$ be a planar graph with a skew-symmetric weight function w on the edges. In the given planar embedding, let the vertices of an inner face f be given by $v_1, v_2, \ldots, v_l$ (clockwise ordering). Then the circulation of the face $w(f)$ is given by $w(v_1, v_2) + w(v_2, v_3) + \cdots + w(v_l, v_1)$.

Instead of assigning the edge weights directly, we will fix circulations for the inner faces of the graph. The following lemma relates the circulations of cycles with that of the faces.

Lemma 7.8 ([BTV09])**.** *In a planar graph with a given planar embedding, the circulation of a cycle in clockwise orientation is the sum of the circulations of the faces inside it.*

Proof sketch. For a cycle C, consider the sum of circulations of all the faces inside it. Any edge which is a part of cycle C contributes to this sum once, as it appears in exactly one of the faces. Also, its direction will be same as that in the clockwise orientation of the cycle. Any edge which lies inside the cycle C, appears in two faces and with opposite directions. Thus, it has zero contribution to the sum. Hence, we get the lemma statement. $\square$

Clearly, fixing positive circulations for all inner faces will avoid any cancellations. [DKR10] gives $+1$ circulation for every face which ensures a nonzero circulation for every cycle. The only task that remains is to assign weights to the edges such that each inner face gets the desired circulation. To do this, we will use the concept of the dual graph.

Definition 7.9 (Dual graph)**.** For a planar graph G and a given planar embedding, its dual graph G^* is a graph that has a vertex for each face of G and any two vertices in G^* are joined by an edge if the corresponding faces in G have a common edge.

Clearly, there is a one-to-one correspondence between the edges of G and G^*. Hence, we treat the edge sets $E(G)$ and $E(G^*)$ of the two graphs as the same set. The next lemma shows that one can achieve arbitrary circulations for the inner faces, using the dual graph of G.

Lemma 7.10 ([Kor09])**.** *Let $G(V, E)$ be a planar graph with F being its set of inner faces in a given planar embedding. For any given function on the inner faces $w' : F \to \mathbb{Z}$, a skew-symmetric weight function $w \colon \vec{E} \to \mathbb{Z}$ on the edges can be constructed in log-space such that each face $f \in F$ has circulation $w'(f)$.*

Proof. The construction in [Kor09] gives +1 circulation to every face of the graph and is in NC. We modify it to assign arbitrary circulations to the faces and argue that it works in log-space.

Let G^* be the dual graph of G and T^* be a spanning tree of G^*. The dual graph can be easily constructed in log-space from the planar embedding, one just needs to identify the edges present in each face. See [NTS95, Rei08] for a log-space construction of a spanning tree. Make the tree T^* rooted at the outer face of G. Let $E(T^*)$ denote the edges of the tree T^* (and also the corresponding edges in graph G). All the edges in $E \setminus E(T^*)$ will get weight 0. For any node f in G^* (a face in G), let T_f^* denote the subtree of T^* rooted at f. Let $w'(T_f^*)$ denote the total sum of the weights in the tree, i.e., $w'(T_f^*) = \sum_{f_1 \in T_f^*} w'(f_1)$. This function can be computed for every node in the tree T^* by the standard log-space tree traversal (see Section 7.1.1). For any inner face f, let e_f be the edge connecting f to its parent in the dual tree T^*. We assign the edge e_f, weight $w'(T_f^*)$ in clockwise direction (w.r.t. face f).

We claim that under this weight assignment, circulation of any inner face f is $w'(f)$. To see this, let us say $f_1, f_2, \ldots, f_k$ are the children of f in the dual tree T^*. These nodes are connected with f using edges $e_{f_1}, e_{f_2}, \ldots, e_{f_k}$ respectively. Now, consider the weights of these edges in the clockwise direction w.r.t. face f. For any $1 \leq i \leq k$, weight of e_{f_i} is $-w'(T_{f_i}^*)$ and weight of e_f is $w'(T_f^*)$. Clearly, sum of all these weights is $w'(f)$. □

This finishes the isolation of a perfect matching in a bipartite planar graph.

7.3 Our Techniques

We start with the idea of Datta, Kulkarni and Roy [DKR10] that a skew-symmetric weight function on the edges such that every cycle has a nonzero circulation implies isolation of a perfect matching in bipartite graphs. To achieve nonzero circulation in a $K_{3,3}$-free or K_5-free graph, we work with its 3-connected or 4-connected component decomposition given by [Wag37, Asa85] which can be constructed in log-space [TW14, STW14]. The components are either planar or constant-sized and share a pair/triplet of vertices. These components form a *tree structure*, when each component is viewed as a node and there is an edge between two components if they share a pair/triplet. For any cycle C in the graph, we break it into its fragments contained within each of these components, which we call *projections* of C. Any such projection can be made into a cycle by adding virtual edges for separating pairs/triplets in the corresponding component.

Circulation of any cycle can be seen as a sum of circulations of its projections. The projections of a cycle can have circulations with opposite signs and thus, can cancel each other. To avoid this cancellation, we observe that the components, where a cycle has a non-empty projection, form a subtree of the component tree. The idea is to assign edge weights using a different scale for each level of nodes in the tree. This ensures that for any subtree, its root node will contribute a weight higher than the total weight from all its other nodes. To avoid any cancellations within a component, weights in a component are given by modifying some known techniques for planar graphs [DKR10, Kor09] and constant sized graphs.

This idea would work only if the component tree has small depth, which might not be true in general. Thus, we create an $O(\log n)$-depth *working tree* by finding 'centers' for the component tree and its subtrees recursively. The working tree 'preserves' the subtree structure in some sense. The construction of such a balanced working tree has been studied in context of evaluating arithmetic expressions [Bre74]. In the literature, this construction is also known as 'centroid decomposition' or 'recursive balanced separators'. Its log-space implementation is more involved.

As the working tree has $O(\log n)$ levels, the straightforward way of using a different scale for each level will lead to edge weights being $n^{O(\log n)}$. So instead, in a component node, we assign weights to only those edges which surround a separating pair/triplet. The weighting scheme ensures that the total weight grows only by a constant multiple, when we move one step higher in the working tree.

Achieving non-zero circulation in log-space also puts directed reachability in UL [RA00, BTV09, TV12]. Thus, we get an alternate proof for the result – directed reachability for $K_{3,3}$-free and K_5-free graphs is in UL [TW14].

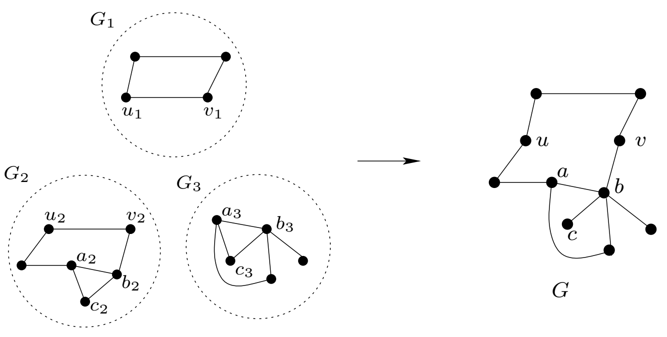

Figure 7.2: Graph G obtained by taking (i) 2-clique-sum of G_1 and G_2 by identifying $\langle u_1, v_1\rangle$ with $\langle u_2, v_2\rangle$ and (ii) 3-clique-sum of the resulting graph with G_3 by identifying $\langle a_2, b_2, c_2\rangle$ with $\langle a_3, b_3, c_3\rangle$.

In Section 7.4, we introduce the concepts of clique-sum, graph decomposition and the corresponding component tree. In Section 7.5, we give a log-space construction of a weight assignment with nonzero circulation for every cycle, for a class of graphs defined via clique-sum operations on planar and constant-sized graphs. In Section 7.6, we argue that $K_{3,3}$-free and K_5-free graphs fall into this class.

7.4 Graph Decomposition

7.4.1 Clique-sum

Clique-sum is a graph operation, via which we will define a special class of graphs. Later, we will see that any $K_{3,3}$-free or K_5-free graph belongs to this class.

Definition 7.11 (Clique-Sum)**.** Let G_1 and G_2 be two graphs each containing a clique (of the same size). A clique-sum of graphs G_1 and G_2 is obtained from their disjoint union by identifying pairs of vertices in these two cliques to form a single shared clique, and by possibly deleting some of the edges in the clique. It is called a k-clique-sum if the cliques involved have at most k vertices.

One can form clique-sums of more than two graphs by a repeated application of clique-sum operation on two graphs (see Figure 7.2). Using this, we define a new class of graphs. Let $\mathcal{P}_c$ be the class of all planar graphs together with all graphs of size at most c, where c is a constant. Define $\langle \mathcal{P}_c\rangle_k$ to be the class of graphs constructed by repeatedly taking k-

clique-sums, starting from the graphs which belong to the class $\mathcal{P}_c$. The starting graphs are called the component graphs. We will construct a nonzero circulation weight assignment for the graphs which belong to the class $\langle \mathcal{P}_c \rangle_3$.

Taking 1-clique-sum of two graphs will result in a graph which is not biconnected. As we are interested in perfect matchings, we only deal with biconnected graphs (see Section 7.1.2). Thus, we assume that every clique-sum operation involves either 2-cliques or 3-cliques. A 2-clique which is involved in a clique-sum operation is called a separating pair. Similarly, a 3-clique is called a separating triplet. In general, they are called separating sets. Note that deletion of any separating pair/triplet will make the graph disconnected.

7.4.2 Component Tree

In general, clique-sum operation can be performed many times using the same separating set. In other words, many components can share a separating set. In Section 7.6, we show that any graph in $\langle \mathcal{P}_c \rangle_3$ can be modified via some matching preserving operations such that on decomposition, any separating set is shared by only two components. Henceforth, in this section we assume this property.

Using this assumption, we can define a component graph for any graph $G \in \langle \mathcal{P}_c \rangle_3$ as follows: each component is represented by a node and two such nodes are connected by an edge if the corresponding components share a separating set. Observe that this component graph is actually a tree. This is because when we take repeated clique-sums, a new component can be attached with only one of the already existing components, as a clique will be contained within one component. In literature [HT73, TW14], the component tree also contains a node for each separating set and it is connected by all the components which share this separating set. But, here we can ignore this node as we have only two sharers for each separating set.

In the component tree, each component is shown with all the separating sets it shares with other components. Thus, a copy of a separating set is present in both its sharer components. Moreover, in each component, a separating set is shown with a virtual clique, i.e., a virtual edge for a separating pair and a virtual triangle for a separating

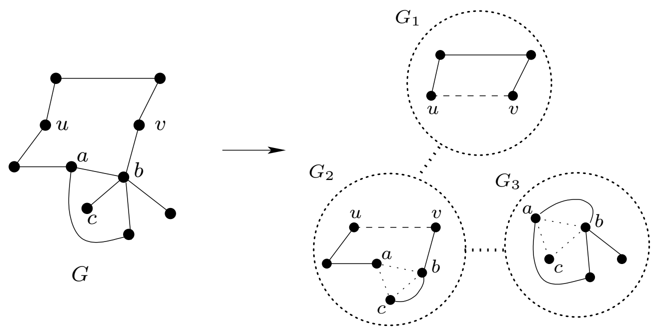

Figure 7.3: A graph $G \in \langle \mathcal{P}_c \rangle_3$ is shown with its component tree. Dotted circles show the nodes and dotted lines connecting them show the edges of the component tree. Dashed lines represent virtual edges and dotted triangles represent the virtual triangles, in the components.

triplet. These virtual cliques represent the paths between the nodes via other components (see Figure 7.3). If any two vertices in a separating set have a real edge in G, then that real edge is drawn in one of the sharing components, parallel to the virtual edge. Note that while a vertex can have its copy in two components, any real edge is present in exactly one component.

In literature [HT73, TW14], for any real edge in a separating set, the component tree contains a new node called "3-bond" (a real edge with two parallel virtual edges). But, here we do not have this node and represent the real edge as mentioned above.

7.5 Nonzero Circulation

In this section, we construct a nonzero circulation weight assignment for a given graph in the class $\langle \mathcal{P}_c \rangle_3$, provided that the component tree and the planar embeddings of the planar components are given. Moreover, to construct this weight assignment we will make some assumptions about the given graph and its component tree.

1. In any component, a vertex is a part of at most one separating set.
2. Each separating set is shared by at most two components.
3. Any virtual triangle in a planar component is always a face (in the given planar embedding).

In Section 7.6, we will show how to construct a component tree for a given $K_{3,3}$-free or K_5-free graph and then to modify it to have these properties. The third property comes naturally, as the inside and outside parts of any virtual triangle can be considered as different components sharing this separating triplet. All these constructions are in log-space. We will not need the bipartiteness assumption in any of these steps. Lemma 7.6 is the only place where bipartiteness is required.

7.5.1 Components of a Cycle

We look at a cycle in the graph as *sum* of many cycles, one from each component the cycle passes through. Intuitively, the original cycle is *broken* at the separating set vertices which were part of the cycle, thereby generating fragments of the cycle in various nodes of the component tree. In all the component nodes containing these fragments, we include the virtual edges of the separating sets in question to complete the fragment into a cycle, thus resulting in component cycles in the component nodes (see Figure 7.4).

Consider a directed cycle $C = \{(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_0)\}$ in a graph $G = (V, E)$. Without loss of generality, consider that G is separated into two components G_1 and G_2 via a separating pair (v_i, v_0) or a separating triplet (v_i, v_0, u), where $1 \leq i < k$ and $u \in V$. Then, one of the components, say G_1, will contain the vertices $v_i, v_{i+1 \bmod k}, \ldots, v_{k-1}, v_0$, and the other ($G_2$) will contain the vertices $v_0, v_1, \ldots, v_{i-1}, v_i$. Then the cycles $C_1 = \{(v_i, v_{i+1 \bmod k}), \ldots, (v_{k-1}, v_0), (v_0, v_i)\}$ and $C_2 = \{(v_0, v_1), \ldots, (v_{i-1}, v_i), (v_i, v_0)\}$ in G_1 and G_2 respectively are the component cycles of C, and we say that C is the sum of C_1 and C_2. Observe that the edges (v_i, v_0) and (v_0, v_i) are virtual.

Repeat the processes recursively for C_1 and C_2 until no separating set breaks a cycle component, and we get the component cycles of the cycle C. Note that any edge in a cycle C is contained in exactly one of its component cycles. Moreover for any component cycle, all its edges, other than the virtual edges, are contained in C.

Observe that for any separating set in a component, a cycle can use one of its vertices to go out of the component and another vertex to come in (this transition is represented by a virtual edge in the component). As any separating set has size at most 3, a cycle can

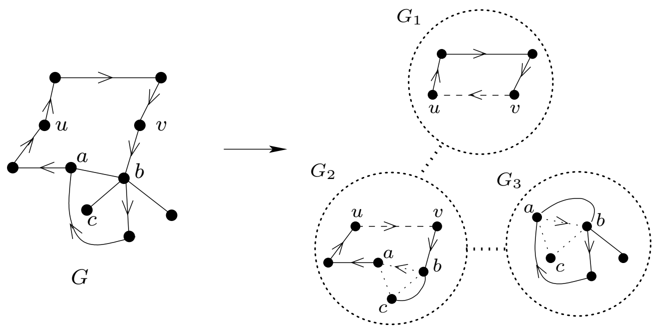

Figure 7.4: Breaking a cycle into its component cycles (projections) in the component tree. Notice that the original cycle and its components share the same set of *real* edges.

visit a node of the component tree only once. In other words, a cycle can have only one component cycle in any component tree node (this would not be true if we had separating sets of size 4). Also, a component cycle can take only one edge of any virtual triangle.

Definition 7.12 (Projection of a cycle)**.** For a given component node N in the component tree, the component cycle of a cycle C in N is called the projection of C on N. If there is no component cycle of C in N, then C is said to have an empty projection on N.

It is easy to see that for any cycle C, the components on which C has a non-empty projection, form a subtree of the component tree. To construct the weight assignment (Section 7.5.2), we will work with the component nodes of the component tree. Within any component, weight of a virtual edge will always be set to zero. Hence, the following lemma.

Lemma 7.13. *The circulation of a cycle is the sum of the circulations of its component cycles.*

Note that for a cycle, its component cycles can have circulations with different signs (positive or negative) as they can have different orientations (clockwise or anti-clockwise) in the planar components. Hence the total circulation can potentially be zero. Our idea is to ensure that one of the component cycles get a circulation greater than all the other component cycles put together. This will imply a nonzero circulation.

7.5.2 Weighting Scheme

The actual weight function we employ is a combination of two weight functions w_0 and w_1. They are combined with an appropriate scaling so that they do not interfere with each other. w_1 ensures that all the cycles which are within one component have a non-zero circulation and w_0 ensures that all the cycles which project on at least two components have a non-zero circulation. We first describe the construction of w_0.

Working Tree: The given component tree can have arbitrary depth, while our weight construction would need the tree-depth to be $O(\log n)$. Thus, we define a new *working tree*. It is a rooted tree which has the same nodes as the component tree, but the edge relations are different. The working tree, in some sense, 'preserves' the subtree structure of the original tree.

For a tree S, its working tree $\mathsf{wt}(S)$ is constructed as follows: Find a 'center' node $c(S)$ in the tree S and mark it as the root of the working tree, $r(\mathsf{wt}(S))$. Deleting the node $c(S)$ from the tree S, would give a set of disjoint trees, say $\{S_1, S_2, \dots, S_k\}$. Apply this procedure recursively on these trees to construct their working trees $\mathsf{wt}(S_1), \mathsf{wt}(S_2), \dots, \mathsf{wt}(S_k)$. Connect each $\mathsf{wt}(S_i)$ to the root $r(\mathsf{wt}(S))$, as a subtree. In other words, make $r(\mathsf{wt}(S_i))$ a child of $r(\mathsf{wt}(S))$ for each S_i. This completes the construction. Let us say the component $c(S)$ shares the separating set τ_i with S_i, then the subtree $\mathsf{wt}(S_i)$ is said to be attached to the root $r(\mathsf{wt}(S))$ at τ_i.

The center nodes are chosen in a balanced way so that the working tree depth is $O(\log n)$. The obvious candidate for a center would be a node whose deletion gives $|S_i| \leq 1/2|S|$ for each S_i. With this definition of center, it is not clear if the whole working tree can be constructed in log-space. Choosing the centers more cleverly, von Braunmühl and Verbeek [vBV83] and later Limaye, Mahajan and Rao [LMR07] gave a log-space construction of such a balanced tree, but in terms of well-matched strings. Das, Datta and Nimbhorkar [DDN13] described a way to translate a tree into a well-matched string and then applied the algorithm of [LMR07] on this string. In Section 7.5.4, we present a direct log-space construction in terms of a tree.

Note that for any two nodes $v_1 \in S_i$ and $v_2 \in S_j$ such that $i \neq j$, $\text{path}(v_1, v_2)$ in S passes through the node $c(S) = r(\mathsf{wt}(S))$. Thus, we get the following property for the working tree.

Claim 7.14. *For any two nodes $u, v \in S$, let their least common ancestor in the working tree $\mathsf{wt}(S)$ be the node a. Then $\text{path}(u, v)$ in the tree S passes through a.*

The root $r(\mathsf{wt}(S))$ of the working tree $\mathsf{wt}(S)$ is said to be at *level* 1. For any other node in $\mathsf{wt}(S)$, its *level* is defined to be one more than the level of its parent. Henceforth, level of a node will always mean its level in the working tree. From Claim 7.14, we get the following.

Claim 7.15. *Let S' be an arbitrary subtree of S, with its set of nodes being $\{v_1, v_2, \ldots, v_k\}$. There exists $i^* \in \{1, 2, \ldots, k\}$ such that for any $j \in [k]$ with $j \neq i^*$, v_j is a descendant of v_{i^*} in the working tree $\mathsf{wt}(S)$.*

Proof. Let l^* be the minimum level of any node in S', and let v_{i^*} be a node in S' with level l^*. We claim that every other node in S' is a descendant of v_{i^*} in the working tree $\mathsf{wt}(S)$. For the sake of contradiction, let there be a node $v_j \in S'$ which is not a descendant of v_{i^*}. Then, the least common ancestor of v_j and v_{i^*} in $\mathsf{wt}(S)$ must have a level, strictly smaller than l^*. By Claim 7.14, this least common ancestor must be present in the tree S'. But, we assumed l^* is the minimum level in S'. Thus, we get a contradiction. □

This claim plays a crucial role in our weight assignment construction, as for any cycle C, the components with a non-empty projection of C form a subtree of the component tree. To assign weights in the graph, we work with the working tree of its component tree. Let the working tree be $\mathcal{T}$. We start by assigning weight to the nodes having the largest level, and move up till we reach level 1, that is, the root node $r(\mathcal{T})$. The idea is that for any cycle C, its unique lowest-level projection should get a circulation higher than the total circulation of all its other projections.

Complementary to the level, we also define *height* of every node in the working tree. Let the maximum level of any node in the working tree be L. Then, the height of a node is defined to be the difference between its level and $L + 1$.

Circulations of cycles spanning multiple components: For any subtree T of the working tree $\mathcal{T}$, the weights to the edges inside the component $r(T)$ will be given by two different schemes depending on whether the corresponding graph is planar or constant sized.

Let the maximum possible number of edges in a constant sized component be m. Then, let K be a constant such that $K > \max(2^{m+2}, 7)$. Also, suppose that the height of a node N is given by the function $h(N)$, and the number of leaves in subtree T is given by $l(T)$. Lastly, suppose the set of subtrees attached at $r(T)$ is $\{T_1, T_2, \ldots, T_k\}$.

Constant sized graph: Let the set of (real) edges of the graph be $\{e_1, e_2, \ldots, e_m\}$. The edge e_j will be given weight $2^j \times K^{h(r(T))-1} \times l(T)$ for an arbitrarily fixed direction. The intuition behind this scheme is that powers of 2 ensure that sum of weights for any subset of edges remain nonzero even when they contribute with different signs.

Planar graph: We work with a given planar embedding of the graph. We use some concepts from Section 7.2. Recall that the circulation of a face is the circulation of the corresponding cycle in the clockwise direction. Instead of directly assigning edge weights, we will fix circulations for the inner faces of the graph. Lemma 7.10 already described how to assign weights to the edges of a planar graph to get the desired circulation for each of the inner faces.

Assigning circulations to the faces: Here, only those inner faces are assigned nonzero circulations which are adjacent to some separating pair/triplet shared with a subtree. This is a crucial idea. As we will see in Lemma 7.16, this ensures that the maximum possible circulation of a cycle grows only by a constant multiple as we move one level down in the working tree.

If T is a singleton, i.e., there are no subtrees attached at T, we give a zero circulation to all the faces (and thus zero weight to all the edges) of $r(T)$. Otherwise, consider a separating pair $\{a, b\}$ where a subtree T_i is attached to $r(T)$. The two faces adjacent to the virtual edge (a, b) will be assigned circulation $2 \times K^{h(r(T_i))} \times l(T_i)$. Similarly, consider a triplet $\{a, b, c\}$ where a subtree T_j is attached. Then all the faces (at most 3) adjacent to the virtual triangle $\{a, b, c\}$ get circulation $2 \times K^{h(r(T_j))} \times l(T_j)$. Repeat this procedure for all the faces adjacent to any pairs and/or triplets where subtrees are attached. If a face is adjacent to more than one virtual edge/triangle, then we just take the sum of different circulations due to each virtual edge/triangle.

Recall that by definition, each face has a positive circulation in the clockwise direction. The intuition behind this scheme is the following: circulation of any cycle in the planar component is just the sum of circulations of the faces inside it (Lemma 7.8). As all of them have the same sign, they cannot cancel each other. Moreover, it will be ensured that the contribution to the circulation from this planar component is higher than the total contribution from all its subtrees, and thus, cannot be canceled.

Now, we formally show that this weighting scheme ensures that all the cycles spanning multiple components in the tree get non-zero circulation.

Nonzero Circulation of a cycle: First, we derive an upper bound on the circulation of any cycle completely contained in a subtree T of the working tree.

Lemma 7.16. *The upper bound on the circulation of any cycle contained in a subtree T of the working tree $\mathcal{T}$ is $U_T = K^{h(r(T))} \times l(T)$.*

Proof. We prove this using induction on the height of $r(T)$.

Base case: The height of $r(T)$ is 1. Notice that this means that $r(T)$ has the maximum level amongst all the nodes in $\mathcal{T}$, and therefore, $r(T)$ is a leaf node, and T is a singleton. Consider the two cases: i)when $r(T)$ is a planar node, and ii)when it is a constant sized node.

By our weight assignment, if $r(T)$ is planar, the total weight of all the edges is zero. On the other hand, if $r(T)$ is a constant sized graph, the maximum circulation of a cycle is the sum of weights of its edges, that is, $\sum_{i=1}^{m}(K^0 \times 1 \times 2^i) < 2^{m+1} \leq K$. Thus, the circulation is upper bounded by $K^{h(r(T))} \times l(T)$ (as $l(T) = 1$).

Induction hypothesis: For any tree T' with $h(r(T')) \leq j-1$, the upper bound is $U_{T'} = K^{h(r(T'))} \times l(T')$.

Induction step: We will prove that for any tree T with $h(r(T)) = j$, the upper bound is $U_T = K^{h(r(T))} \times l(T)$.

Let the subtrees attached at $r(T)$ be $\{T_1, T_2, \ldots, T_k\}$. For any cycle in T, sum of the circulations of its projections on the subtrees $T_1, T_2, \ldots, T_k$ can be at most $\sum_{i=1}^{k} U_{T_i}$.

First, we handle the case when $r(T)$ is planar. For any subtree T_i, the total circulation of faces in $r(T)$ due to connection to T_i can be $6 \times K^{h(r(T_i))} \times l(T_i)$. This is because the circulation of each face adjacent to the separating set connecting with T_i is $2 \times K^{h(r(T_i))} \times l(T_i)$, and there can be at most 3 such faces. Thus,

$$\begin{aligned}
U_T &= \sum_{i=1}^{k} U_{T_i} + \sum_{i=1}^{k} \Big(6 \times K^{h(r(T_i))} \times l(T_i)\Big) \\
&= \sum_{i=1}^{k} \Big(K^{h(r(T_i))} \times l(T_i)\Big) + \sum_{i=1}^{k} \Big(6 \times K^{h(r(T_i))} \times l(T_i)\Big) \\
&= 7 \times K^{h(r(T))-1} \times \sum_{i=1}^{k} l(T_i) && (\because \forall i,\ h(r(T_i)) = h(r(T)) - 1) \\
&< K^{h(r(T))} \times \sum_{i=1}^{k} l(T_i) && (\because K > 7) \\
&= K^{h(r(T))} \times l(T)
\end{aligned}$$

Now, consider the case when $r(T)$ is a small non-planar graph. The maximum possible contribution from edges of $r(T)$ to the circulation of a cycle in T is less than $2^{m+1} \times K^{h(r(T))-1} \times l(T)$. Similar to the case when $r(T)$ is planar, contribution from all subtrees is at most $K^{h(r(T))-1} \times l(T)$. The total circulation of a cycle in T can be at most the sum of these two bounds, and is thus bounded above by $(2^{m+1} + 1) \times K^{h(r(T))-1} \times l(T)$. Since, $K > 2^{m+2}$, the total possible circulation is less than $K^{h(r(T))} \times l(T)$.

Therefore, the upper bound $U_T = K^{h(r(T))} \times l(T)$. $\square$

To see that each cycle gets a nonzero circulation, recall Lemma 7.13, which says that the circulation of the cycle is the sum of circulations of its projections on different components. Consider a cycle C. Recall that components with a non-empty projection of C form a subtree S_C in the component tree. From Claim 7.15, we can find a node $v^* \in S_C$ such that all other nodes in S_C are its descendants in the working tree $\mathcal{T}$. Thus, v^* is the unique minimum level component on which C has a non-empty projection. Now, we show two things: (i) the contribution to the circulation from this component is nonzero, and (ii)it is larger than sum of all the circulation contributions from all its subtrees in the working tree.

Let v^* be the root of a subtree T in the working tree. Let the subtrees attached at $r(T)$ $(= v^*)$ be $\{T_1, T_2, \ldots, T_k\}$ and the separating sets in $r(T)$ at which they are attached be $\{\tau_1, \tau_2, \ldots, \tau_k\}$ respectively.

Case 1: when $r(T)$ is a constant-sized component. It is easy to see that the circulation of any cycle in this component will be nonzero as long as it takes a real edge, because the weights given are powers of 2. Also, the minimum weight of any edge in $r(T)$ is $2K^{h(r(T))-1} \times l(T) = 2 \times \sum_{i=1}^{k} U_{T_i}$. Thus, when a cycle takes a real edge, contribution to its circulation from $r(T)$ is larger than the contribution from higher level components (components in the subtrees attached at $r(T)$). Further, any cycle has to take a real edge, as the virtual edges and triangles all have disjoint set of vertices. (Here, the virtual triangle does not count as a cycle).

Case 2: when $r(T)$ is a planar component. The crucial observation here is that in a planar graph, all the faces inside a cycle contribute to its circulation in the same orientation (Lemma 7.8).

Since C passes through at least one of the subtrees attached at $r(T)$, say T_i, it must go through the separating set τ_i. Hence, the projection of C in $r(T)$, say C', must use the virtual edge (or one of the edges in the virtual triangle) corresponding to τ_i. This would imply that at least one of the faces adjacent to τ_i is inside C'. This is true for any subtree T_i which C passes through. As the faces adjacent to separating sets have nonzero circulations and each face has a positive circulation in clockwise direction, the circulation of C' is nonzero.

Recall that circulation of any face adjacent to τ_i is $2U_{T_i}$, where U_{T_i} is the upper bound on circulation contribution from T_i. This implies that the circulation of C' will surpass the total circulation from all the subtrees which C passes through. Thus, we can conclude the following.

Lemma 7.17. *Circulation of any cycle which passes through at least two components is nonzero.*

Weights from faces to edges: Now, we come back to the question of assigning weights to the edges in a planar component such that the faces get the desired circulations. Lemma 7.10 describes this procedure for any planar graph.

The scheme in Lemma 7.10 can assign a weight to any edge, while we are not allowed to give weights to virtual edges/triangles. So, first we collapse all the virtual triangles to one node and all the virtual edges to one node. As no two virtual triangles/edges are adjacent, after this operation, every face remains a non-trivial face (except the virtual triangle face). Now, we apply the procedure from Lemma 7.10. After undoing the collapse, the circulations of the faces will not change and we will have the desired circulations.

Circulation of cycles contained within a single component: To construct w_1 for planar components, we assign $+1$ circulation to every face using Lemma 7.10 (similar to the case of multiple components). This would ensure nonzero circulation for every cycle within the planar component. This construction has been used in [Kor09] for bipartite planar graphs. Tewari and Vinodchandran [TV12] also give a log-space construction which ensures nonzero circulation for all cycles in a planar graph, using Green's theorem.

For the non-planar components, w_0 already ensures that each cycle has non-zero circulation. Therefore, we set $w_1 = 0$. Use a linear combination of w_0 and w_1 such that they do not interfere with each other. This together with Lemma 7.17 gives us the following.

Lemma 7.18. *Circulation of any cycle is non-zero.*

Polynomially bounded weights: Now, we show that the weight given by this scheme is polynomially bounded.

Lemma 7.19. *The total weight given by the weighting scheme is polynomially bounded.*

Proof. The weight w_1 is polynomially bounded according to the procedure in Lemma 7.10.

Consider w_0. Observe that the upper bound $U_{\mathcal{T}}$ for the circulation of a cycle in $\mathcal{T}$ is actually just the sum of weights of all the edges in constant sized components, and of all the faces in planar components. By the construction given in the proof of Lemma 7.10, weight of any edge in the planar component is bounded by the sum of circulations of all the faces. Therefore, $U_{\mathcal{T}}$ gives the bound on the weight function w_0. Since the maximum level of any node in $\mathcal{T}$ can be at most $O(\log|\mathcal{T}|)$, the height of $r(T)$, that is $h(r(T)) = O(\log|\mathcal{T}|)$. Also, the total number of leaves in $\mathcal{T}$ is at most $|\mathcal{T}|$.

$$U_{\mathcal{T}} = K^{h(r(\mathcal{T}))} \times l(\mathcal{T}) \leq K^{O(\log|\mathcal{T}|)} \times |\mathcal{T}| = |\mathcal{T}|^{O(\log K)}|\mathcal{T}| = |\mathcal{T}|^{O(\log K)}$$

If n is the size of the original graph G, then clearly $|\mathcal{T}| \leq n$. Therefore, $U_{\mathcal{T}} = O(n^{O(\log K)})$. Recall that K is a constant, and thus, w_0 is also polynomially bounded.

Since we use a linear combination of w_0 and w_1, the total weight function is polynomially bounded. □

7.5.3 Complexity of the Weight Assignment

Section 7.5.4 gives a log-space construction of the working tree. We use simple log-space procedures in sequence to assign the weights in the working tree. After construction of the working tree, we use iterative log-space procedures to store the following for each node: i) the level of the node, and ii) the number of leaves in the subtree rooted at it. Both just require a tree traversal while keeping a counter, and can clearly be done in log-space (see Section 7.1.1). We use another straightforward log-space function to compute height of every node using the maximum level amongst all the nodes. For each component node of the working tree, we store a list of all the separating sets in it and corresponding pointers for the subtrees attached at them.

Next, we iterate on the nodes of the working tree to assign the weights. For every non-planar component N, we assign a weight of $2^i \times K^{(h(N)-1)} \times l(T(N))$ to the i-th edge of component N, where $T(N)$ is the subtree rooted at N.

For every planar component N, we visit all its virtual edges/triangles. For a given virtual edge/triangle τ_i, let T_i be the subtree attached to N at τ_i. We add a circulation of $2 \times K^{h(r(T_i))} \times l(T_i)$ to all the faces adjacent to τ_i. Clearly, the procedure works in log-space. As the last step, we find the weights for the edges which would give the desired circulations of the faces. Lemma 7.10 shows that it can be done in log-space.

7.5.4 Construction of the Working Tree

Now, we describe a log-space construction of the working tree. The idea is obtained from the construction of [LMR07, Lemma 6], where they create a $O(\log n)$-depth tree of well-matched substrings of a given well-matched string. Recall that for a tree S, the working tree $\mathsf{wt}(S)$ is constructed by first choosing a center node $c(S)$ of S and marking it as the root of $\mathsf{wt}(S)$, and then recursively finding the working trees for each component obtained by removing the node $c(S)$ from S and connecting them to the root of $\mathsf{wt}(S)$, as subtrees.

First, consider the following possible definition of the center: for any tree S with n nodes, one can define its center to be a node whose removal would give disjoint components of size $\leq 1/2|S|$. Finding such a center is an easy task and can be done in log-space. Clearly, the depth of the working tree would be $O(\log n)$. It is not clear whether the recursive procedure of finding centers for each resulting component can be done in log-space. Therefore, we give a more involved way of defining centers, so that the whole recursive procedure can be done in log-space.

First, we make the tree S rooted at an arbitrary node r (see Section 7.1.1). For any node v, let S_v denote the subtree of S, rooted at v. For any node v and one of its descendant nodes v' in S, let $S_{v,v'}$ denote the tree $S_v \setminus S_{v'}$. Moreover $S_{v,\epsilon}$ would just mean S_v, for any v. With our new definition of the center, at any stage of the recursive procedure, the component under consideration will always be of the form $S_{v,v'}$, for some nodes $v, v' \in S$. Now, we give a definition of the center for a rooted tree of the form $S_{v,v'}$.

Center $c(S_{v,v'})$**:** case (i) When $v' = \epsilon$, i.e., the given tree is S_v. Let c be a node in S_v, such that its removal gives components of size $\leq 1/2|S_v|$. If there are more than one such nodes then choose the lexicographically smaller one (there is at least one such center [Jor69]). Define c as the center of $S_{v,v'}$.

Let the children of c in S_v be $\{c_1, c_2, \ldots, c_k\}$. Clearly, after removing c from S_v, the components we get are $S_{c_1}, S_{c_2}, \ldots, S_{c_k}$ and $S_{v,c}$. Thus, they are all of the desired form and have size $\leq 1/2|S_v|$.

case (ii) When v' is an actual node in S_v. Let the node sequence on the path connecting v and v' be $(u_0, u_1, \ldots, u_p)$, with $u_0 = v$ and $u_p = v'$. Let $0 \leq i < p$ be the least index such that $|S_{u_{i+1},v'}| \leq 1/2|S_{v,v'}|$. This index exists because $|S_{u_p,v'}| = 0$. Define u_i as the center of $S_{v,v'}$.

Let the children of u_i, apart from u_{i+1}, be $\{c_1, c_2, \ldots, c_k\}$. After removal of u_i from $S_{v,v'}$, the components we get are $S_{c_1}, S_{c_2}, \ldots, S_{c_k}$, $S_{u_{i+1},v'}$ and S_{v,u_i}. By the choice of i, $|S_{u_i,v'}| > 1/2|S_{v,v'}|$. Thus, $|S_{v,u_i}| \leq 1/2|S_{v,v'}|$. So, the only components for which we do not have a guarantee on their sizes, are $S_{c_1}, S_{c_2}, \ldots, S_{c_k}$. Observe that when we find a center for the tree $S_{c_j,\epsilon}$ in the next recursive call, it will fall into case (i) and the components we get will have their sizes reduced by a factor of $1/2$.

Thus, we can conclude that in the recursive procedure for constructing the working tree, we reduce the size of the component by half in at most two recursive calls. Hence, the depth of working tree is $O(\log n)$. Now, we describe a log-space procedure to construct the working tree.

Lemma 7.20. *For any tree S, its working tree $\mathsf{wt}(S)$ can be constructed in log-space.*

Proof. We just describe a log-space procedure for finding the parent of a given node x in the working tree. Running this procedure for every node will give us the working tree.

Find the center of the tree S. Removing the center would give many components. Find the component S_1, to which the node x belongs. Apply the same procedure recursively on S_1. Keep going to smaller components which contain x, till x becomes the center of some component. The center of the previous component in the recursion will be the parent of x in the working tree.

In this recursive procedure, to store the current component $S_{v,v'}$, we just need to store two nodes v and v'. Apart from these, we need to store center of the previous component and size of the current component.

To find the center of a given component $S_{v,v'}$, go over all possibilities of the center, depending on whether v' is ϵ or a node. For any candidate center c, find the sizes of the

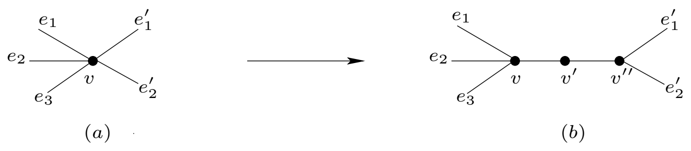

Figure 7.5: Vertex-split: A vertex v is split into three vertices v, v', v'', which are connected by a path. Some of the edges incident on v are transferred to v''.

components generated if c is removed. Check if the sizes satisfy the specified requirements. Any of these components is also of the form $S_{u,u'}$ and thus can be stored with two nodes.

By the standard log-space traversal of a tree (see Section 7.1.1), for any given tree $S_{v,v'}$, one can count the number of nodes in it and test membership of a given node. Thus, the whole procedure works in log-space. □

7.6 $K_{3,3}$-free and K_5-free Graphs

In this section, we will see that any $K_{3,3}$-free or K_5-free graph falls into the class $\langle \mathcal{P}_c \rangle_3$. We will show how to construct the desired component tree for any given $K_{3,3}$-free or K_5-free graph and modify it to satisfy the assumptions made in Section 7.5. All these constructions are in log-space. As mentioned before, we do not need to assume bipartiteness.

7.6.1 Matching Preserving Operation

Vertex-split: For a graph G, we define an operation called *vertex-split*, which *preserves matchings*, as follows: Let v be a vertex and let X be the set of all edges incident on v. Let $X_1 \sqcup X_2$ be an arbitrary partition of X. Create two new vertices v' and v'' (see Figure 7.5). Make the edges (v, v') and (v', v''). We call these two edges as *auxiliary edges*. For all the edges in X_2, change their endpoint v to v''. We denote this operation by vertex-split(v, X_1, X_2).

Let the modified graph be G'. One can go back to the graph G by identifying vertices v, v' and v'' and deleting auxiliary edges. This operation is *matching preserving* in the following sense.

Lemma 7.21. *There is a one-one correspondence between perfect matchings of G and G'.*

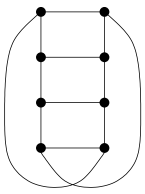

Figure 7.6: The four-rung Möbius ladder V_8.

Proof. Consider a perfect matching M in G, where v is matched with a vertex in X_1. It is easy to see that the matching $M' := M \cup \{(v', v'')\}$ is a perfect matching in G'. The other case when v is matched with a vertex in X_2 is similar.

Consider a perfect matching M' in G'. Removing the auxiliary edge from M' and identifying the vertices v, v' and v'' will give us a perfect matching in G. $\square$

7.6.2 Component Tree

Wagner [Wag37] and Asano [Asa85] gave exact characterizations of K_5-free graphs and $K_{3,3}$-free graphs, respectively. These characterizations essentially mean that any graph in these two classes can be constructed by taking 3-clique-sums of graphs which are either planar or have size bounded by 8. Recall that $\langle \mathcal{C} \rangle_k$ denotes the class of graphs obtained by taking repeated k-clique-sums of graphs starting from the graphs in class $\mathcal{C}$.

Theorem 7.22 ([Asa85])**.** *Let $\mathcal{C}$ be the class of all planar graphs together with the 5-vertex clique K_5. Then $\langle \mathcal{C} \rangle_2$ is the class of $K_{3,3}$-free graphs.*

Theorem 7.23 ([Wag37, Khu88])**.** *Let $\mathcal{C}$ be the class of all planar graphs together with the four-rung Möbius ladder V_8 (Figure 7.6). Then $\langle \mathcal{C} \rangle_3$ is the class of K_5-free graphs.*

As mentioned in Section 7.1.2, we can assume that the given graph is biconnected. It is known that for any given biconnected $K_{3,3}$-free graph G, its component tree can be constructed in log-space [TW14, Lemma 3.8]. The components here are all planar or K_5, which share separating pairs. Also, for any given biconnected K_5-free graph G, its component tree can be constructed in log-space [STW14, Definition 5.2, Lemma 5.3]. The components here are all planar or V_8. They can share a separating pair or a separating triplet.

The procedure of [TW14, STW14] for computing the component tree goes on the following lines: As the graph is assumed to be biconnected, there are no articulation points. First step is to find all separating pairs and triplets. To do this, go over each pair and triplet of vertices and test whether its deletion makes the graph disconnected. To find the components of the component tree, go over all pairs of vertices (u_1, u_2): u_1 and u_2 will be in different components if after deletion of some separating pair/triplet, there is no path between u_1 and u_2. Otherwise they will be in the same component of the component tree. Note that it is possible that u_1, u_2 belong to the same component and u_2, u_3 belong to the same component but u_1, u_3 do not. It is easy to find which components share a separating pair/triplet with each other. There is a special treatment for the components which are just cycles, because in a cycle, almost every pair is a separating pair. They keep the cycle as one component instead of splitting it at every separating pair.

As connectivity can be tested in log-space [Rei08], the above procedure works in log-space. Moreover, a planar embedding of a planar component can be computed in log-space [AM04, Rei08].

The component tree defined in [TW14, STW14] slightly differs from our definition in Section 7.4.2. They have an extra component for each separating set. This component is connected to all the components which share this separating set. Moreover, whenever there is a real edge between two nodes of a separating set, it is represented by a 3-bond component (one real edge and two parallel virtual edges). The 3-bond component is also connected to the corresponding separating set node. For our purposes, these two kinds of components are not needed.

For any given biconnected $K_{3,3}$-free graph or K_5-free graph G, we start with the component tree which is constructed by [TW14, STW14]. We show how to modify the component tree, in log-space, to have the assumptions made in Section 7.5.

Applying the clique-sum operations on the modified component tree will give us the actual modified graph G'. We will argue that all these modifications in G are just repeated application of the vertex-split operation (Lemma 7.21) in G. Thus, these are matching preserving. As mentioned earlier, from a perfect matching in G', one can get a perfect matching in G by just deleting the auxiliary vertices and edges created in the vertex-split operations.

We emphasize here that these operations might create some new pairs/triplets in the graph G' such that their removal will make the graph disconnected. But, we do not consider these new pairs/triplets as separating sets. By a separating pair/triplet we only mean those pairs/triplets which are shared by different components in the original decomposition of G.

(i) Removing "3-bond" components: For all the 3-bond components we do the following: Remove the 3-bond component. Let τ be the separating set and C_τ be the corresponding node in the component tree, where this 3-bond component is attached (a 3-bond component is always a leaf). Take an arbitrary component attached to C_τ. This component will have a virtual clique for τ. Make an appropriate real edge parallel to the existing virtual edge, in this virtual clique corresponding to τ. Note that if this component was planar, it will remain so. Moreover, it is easy to adjust the planar embedding. Clearly, this operation can be done in log-space. This does not change the actual graph G in any way.

(ii) Any separating set is shared by at most two components: Let τ be a separating set shared by m components $G_1, G_2, \ldots, G_m$. Let the cardinality of τ be t (t can be 2 or 3). Let us define a gadget M as follows: it has three sets of nodes $\{a_i \mid 1 \leq i \leq t\}$, $\{b_i \mid 1 \leq i \leq t\}$, $\{c_i \mid 1 \leq i \leq t\}$. For each i, connect a_i with b_i by a length-2 path and also connect a_i with c_i by a length-2 path. Make 3 virtual cliques each of size t, one each for nodes $\{a_i\}_i$, $\{b_i\}_i$ and $\{c_i\}_i$. Thus, three components can be attached with M.

Now, we construct a binary tree T which has exactly $m-1$ leaves. Replace leaves of T with components $G_2, G_3, \ldots, G_m$. Replace all other nodes of T with copies of the gadget M. Further, make an edge between component G_1 and the root of T (see Figure 7.7). In this binary tree, any node of type M (gadget) shares its separating set $\{a_i\}_i$ with its parent

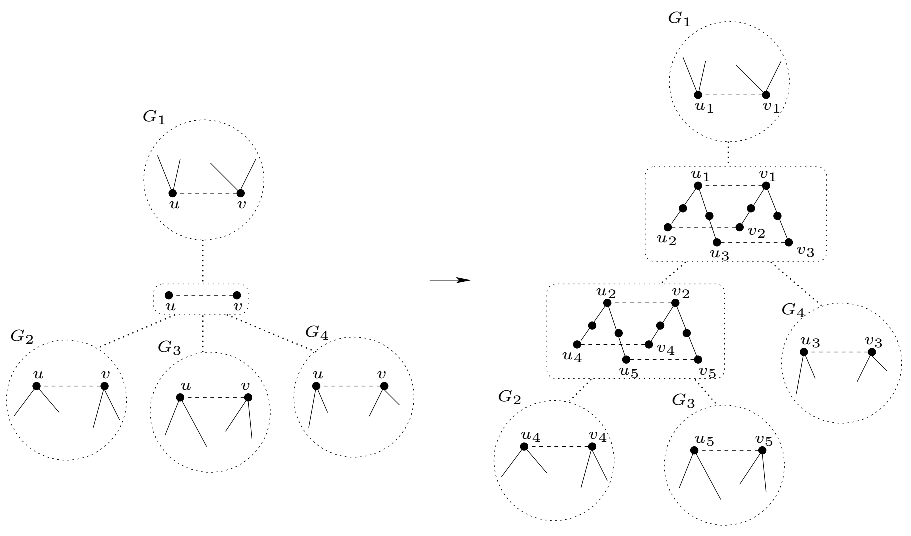

Figure 7.7: (a) A separating pair $\langle u, v\rangle$ is shared by four components G_1, G_2, G_3, G_4. (b) Copies of $\langle u, v\rangle$ connected by length-2 paths, to form a binary tree. Different copies are shared by different components.

node, $\{b_i\}_i$ with its left child node and $\{c_i\}_i$ with its right child node. The components $G_2, G_3, \ldots, G_m$ share their copy of τ with their respective parent nodes in the tree T. The component G_1 shares its copy of τ with the root node of T.

Doing this procedure for every separating set will ensure that every separating set is shared between at most two components. Moreover, now there is no extra component for the separating set, and the components which share a separating set are joined directly by an edge. A binary tree with $m-1$ leaves can be easily constructed in log-space (Take nodes $\{x_1, x_2, \ldots, x_{2m-3}\}$, x_i has children x_{2i} and x_{2i+1}). All the other operations here are local like deleting and creating edges and changing vertex labels. Thus it can be done in log-space.

Now, we want to argue that this operation is matching preserving for the actual graph G. Let us view this operation as a repeated application of the following operation: Partition the set of components $\{G_2, G_3, \ldots G_m\}$ in two parts, say G_1' and G_1''. Now, take a copy of the gadget M and connect it to all three components G_1, G_1' and G_1''. M shares its separating sets $\{a_i\}_i$, $\{b_i\}_i$ and $\{c_i\}_i$ with G_1, G_1' and G_1'' respectively. In the actual graph G, this operation separates the edges incident on a vertex in τ into three parts: edges from G_1, G_1' and G_1'' respectively. These three sets of edges are now incident on

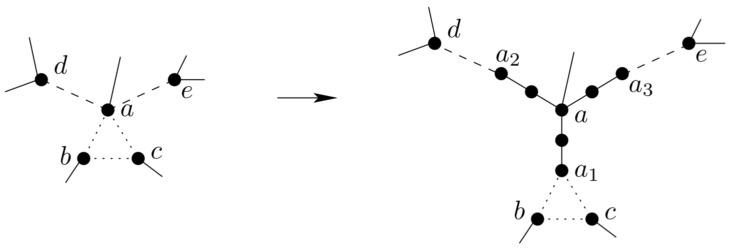

Figure 7.8: (a) Vertex a is a part of two separating pairs $\langle a, d\rangle$ and $\langle a, e\rangle$ and a separating triplet $\langle a, b, c\rangle$. (b) Vertex-Split is applied on vertex a, 3 times, to split it into a star. The new separating sets are $\langle a_1, b, c\rangle$, $\langle a_2, d\rangle$ and $\langle a_3, e\rangle$.

three different copies of the vertex. Moreover, the first copy is connected to the other two copies via a length-2 path. Hence, it is easy to see this as applying vertex-split operation (Lemma 7.21) twice. Now, we recursively do the same operation after partitioning the set of components G_1' and G_1'' further. Thus, the whole operation can be seen as a vertex-split operation applied many times in the actual graph G.

Instead of a binary tree we could have also taken a tree with one root and $m-1$ leaves. This operation would also be matching preserving but the component size will depend on m. On the other hand, in our construction the new components created have size at most 15 (number of real edges is bounded by 12). Thus, the graph G' remains in class $\langle \mathcal{P}_c\rangle_3$.

(iii) Any vertex is a part of at most one separating set: Let a be vertex in a component C, where it is a part of separating sets $\tau_1, \tau_2, \ldots, \tau_m$. We apply the vertex-split operation (Lemma 7.21) on a, m times, to split a into a star. Formally, create a set of m new nodes $a_1, a_2, \ldots, a_m$. Connect each a_i with a by a path of length 2. For each i, replace a with a_i in the separating set τ_i. Let the updated separating set be τ_i'. The edge in the component tree which corresponds to τ_i, should now correspond to τ_i'. Any real edge in the component C which is incident on a, remains that way (see Figure 7.8). Clearly, doing this for every vertex in all the components will ensure that every vertex is a part of at most one separating set.

It is easy to see that a planar component will remain planar after this operation. The modification of the planar embedding and other changes here are local and can be done in log-space.

Now, we want to argue that this operation is matching preserving. Let us see how does this operation modifies the actual graph G. Let C_i be the component which shares τ_i with C. Removal of τ_i would split the graph G into two components, say G'_i and G''_i, where G'_i is the one containing C. The above operation means that any edge in G''_i which was incident on a, is now incident on a_i instead of a. As each a_i is connected to a by a length-2 path, this operation can be seen as a repeated application of the vertex-split operation (Lemma 7.21). Thus, this operation is matching preserving.

Increase in the size of non-planar components: After this operation, the size of each component will grow. Let us find out the new bound on the size of constant-sized graphs. For a $K_{3,3}$-free graph, all non-planar components are of type K_5. Moreover, they are only involved in a 2-clique-sum. Hence, it can have at most $\binom{5}{2} = 10$ separating pairs. In this case, each vertex is a part of four separating pairs. Thus, each vertex will be split into a 4-star, creating 8 new vertices and 8 new edges. Totally, there will be 45 vertices and 40 real edges. Additionally, there can be some already existing real edges, at most 10. Thus, the total number of edges is bounded by 50.

For a K_5-free graph, all the non-planar components are of type V_8. Note that V_8 does not have a 3-clique, thus, can only be be involved in a 2-clique-sum. In the worst case, it will have 12 separating pairs. Each vertex will be a part of 3 separating pairs. Hence, each vertex will be split into a 3-star, creating 6 new vertices and 6 new edges. Totally, there will be 56 vertices and 48 edges. Thus, together with already existing real edges, total number of real edges is bounded by 60.

(iv) A separating triplet in a planar component already forms a face: If a separating triplet does not form a face in a planar component, then the two parts of the graph – one inside the triplet and the other outside – can be considered as different components sharing this triplet. In fact, the construction in [STW14] already does this. When they decompose a graph with respect to a triplet, the different components one gets by deleting this triplet are all considered different components in the component tree.

7.7 Discussion

One of the open problems is to construct an isolating weight assignment for a more general class of graphs, in particular, for all bipartite graphs. Note that *nonzero circulation* for every cycle is sufficient but not necessary for constructing an isolating weight assignment. Although existence of an isolating weight assignment can be shown by randomized arguments, no such arguments exist for showing the existence of a nonzero circulation weight assignment. It needs to be investigated whether it is possible to achieve a nonzero circulation for every cycle (with polynomially bounded weights) in a complete bipartite graph. Log-space construction of such a weight assignment would imply that Bipartite Perfect Matching is in NC and answer the NL=UL? question.

Like $K_{3,3}$-free or K_5-free graphs, small genus graphs are another generalization of planar graphs for which COUNT-PM is in NC [GL99, KMV08]. Thus, SEARCH-PM for small genus bipartite graphs is in NC [KMV08]. Can we do a log-space isolation of a perfect matching for these graphs?

The isolation question is also open for general planar graphs. In fact, planar graphs do not have any NC algorithm for SEARCH-PM, via isolation or otherwise. On the other hand, counting the number of perfect matchings in a planar graph is in NC. It is surprising, as counting seems to be a harder problem than isolation.

Chapter 8

Conclusion

This thesis makes an incremental progress on the derandomization question for two problems: polynomial identity testing and parallel algorithms for matching.

For polynomial identity testing, one of our main contributions is the concept of *basis isolation* (Chapter 3). We show that basis isolation for polynomial D over an algebra implies isolation of a monomial in the corresponding polynomial $C = \mathbf{1} \cdot D$ over the field. As seen in Chapter 5, basis isolation not only implies a hitting-set but also gives low-support rank-concentration. An open question is whether we can do basis isolation for ROABPs in polynomial time. To do this, one must avoid the divide and conquer approach (Lemma 3.6). Another question is to find if the basis isolation approach can work for other circuit models, especially for depth-3 multilinear circuits.

Another novelty we add is the use of a *polynomial map* to get a hitting-set for known-order ROABP (Chapter 5). In all the previous approaches on ROABPs and set-multilinear circuits, the final map used might have been a polynomial map but at the lowest level, the basic map used was a monomial map, i.e., a variable is mapped to a univariate monomial.

We have made significant progress on ROABPs and its special cases, but a polynomial time hitting-set is still elusive. The following are the simplest models for which finding a polynomial time hitting-set is open.

- Read-once formulas

- Diagonal circuits
- Basic set-multilinear circuits ($\sum_i b_i \prod_j (1 + a_{ij} x_i)$)

As discussed before, our result on ROABPs with known variable order was inspired from one of the PRG constructions for ROBPs. It would be interesting to find more concrete connections between hitting-sets for ROABPs and PRGs for ROBPs.

For bipartite matching, we made some progress on derandomizing the isolation lemma. We showed that the approach of nonzero circulation, which was used for bipartite planar graphs, also works for $K_{3,3}$-free or K_5-free bipartite graphs. The next class of graphs, for which one can hope that the same approach would work, is $O(\log n)$-genus bipartite graphs.

The big open question is to isolate a matching in a bipartite graph. It is not at all clear if a nonzero circulation for each cycle can be achieved in a general bipartite graph. It would be interesting to show that this is not possible for the complete bipartite graph.

Bibliography

[AB03] Manindra Agrawal and Somenath Biswas. Primality and identity testing via Chinese remaindering. *J. ACM*, 50(4):429–443, July 2003.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[AGGT14] Rahul Arora, Ashu Gupta, Rohit Gurjar, and Raghunath Tewari. Derandomizing isolation lemma for $k_{3,3}$-free and k_5-free bipartite graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:161, 2014.

[AGKS13] Manindra Agrawal, Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Hitting-sets for low-distance multilinear depth-3. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:174, 2013.

[AGKS15] Manindra Agrawal, Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Hitting-sets for ROABP and sum of set-multilinear circuits. *SIAM J. Comput.*, 44(3):669–697, 2015.

[Agr05] Manindra Agrawal. Proving lower bounds via pseudo-random generators. In *FSTTCS*, volume 3821 of *Lecture Notes in Computer Science*, pages 92–105, 2005.

[AHT07] Manindra Agrawal, Thanh Minh Hoang, and Thomas Thierauf. The polynomially bounded perfect matching problem is in NC^2. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007*, volume 4393 of *Lecture Notes in Computer Science*, pages 489–499. Springer Berlin Heidelberg, 2007.

[AKS02] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in P. *Annals of Mathematics*, 2:781–793, 2002.

[AM04] Eric Allender and Meena Mahajan. The complexity of planarity testing. *Information and Computation*, 189(1):117 – 134, 2004.

[AM08] Vikraman Arvind and Partha Mukhopadhyay. Derandomizing the isolation lemma and lower bounds for circuit size. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008, Boston, MA, USA, August 25-27, 2008. Proceedings*, pages 276–289, 2008.

[ARZ99] Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *J. Comput. Syst. Sci.*, 59(2):164–181, 1999.

[Asa85] Takao Asano. An approach to the subgraph homeomorphism problem. *Theoretical Computer Science*, 38(0):249 – 267, 1985.

[ASS13] Manindra Agrawal, Chandan Saha, and Nitin Saxena. Quasi-polynomial hitting-set for set-depth-formulas. In *STOC*, pages 321–330, 2013.

[AV08] Manindra Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 67–75, 2008.

[BDVY13] Andrej Bogdanov, Zeev Dvir, Elad Verbin, and Amir Yehudayoff. Pseudorandomness for width-2 branching programs. *Theory of Computing*, 9:283–293, 2013.

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

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

[BOT88] Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 301–309, New York, NY, USA, 1988. ACM.

[Bre74] Richard P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, April 1974.

[Bry86] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, August 1986.

[BTV09] Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Trans. Comput. Theory*, 1(1):4:1–4:17, February 2009.

[DDN13] Bireswar Das, Samir Datta, and Prajakta Nimbhorkar. Log-space algorithms for paths and matchings in k-trees. *Theory of Computing Systems*, 53(4):669–689, 2013.

[De11] Anindya De. Pseudorandomness for permutation and regular branching programs. In *IEEE Conference on Computational Complexity*, pages 221–231, 2011.

[DK98] Elias Dahlhaus and Marek Karpinski. Matching and multidimensional matching in chordal and strongly chordal graphs. *Discrete Applied Mathematics*, 84(13):79 – 91, 1998.

[DKR10] Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47:737–757, 2010.

[DKTV12] Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N. V. Vinodchandran. Space complexity of perfect matching in bounded genus bipartite graphs. *J. Comput. Syst. Sci.*, 78(3):765–779, 2012.

[DL78] Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193 – 195, 1978.

[DNTW09] Samir Datta, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Graph isomorphism for $K_{3,3}$-free and K_5-free graphs is in log-space. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, pages 145–156, 2009.

[dOSV15] Rafael Mendes de Oliveira, Amir Shpilka, and Ben Lee Volk. Subexponential size hitting sets for bounded depth multilinear formulas. In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, pages 304–322, 2015.

[DS07] Zeev Dvir and Amir Shpilka. Locally decodable codes with two queries and polynomial identity testing for depth 3 circuits. *SIAM J. Comput.*, 36(5):1404–1434, 2007.

[DSY09] Zeev Dvir, Amir Shpilka, and Amir Yehudayoff. Hardness-randomness tradeoffs for bounded depth arithmetic circuits. *SIAM Journal of Computing*, 39(4):1279–1293, 2009.

[Edm65] Jack Edmonds. Path, trees, and flowers. *Canadian J. Math.*, 17:449467, 1965.

[FS12] Michael A. Forbes and Amir Shpilka. On identity testing of tensors, low-rank recovery and compressed sensing. In *STOC*, pages 163–172, 2012.

[FS13a] Michael A. Forbes and Amir Shpilka. Explicit noether normalization for simultaneous conjugation via polynomial identity testing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, pages 527–542, 2013.

[FS13b] Michael A. Forbes and Amir Shpilka. Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. In *FOCS*, pages 243–252, 2013.

[FSS14] Michael A. Forbes, Ramprasad Saptharishi, and Amir Shpilka. Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 867–875, 2014.

[GK87] Dima Grigoriev and Marek Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC (extended abstract). In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 166–172, 1987.

[GKKS13] Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth three. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 578–587, 2013.

[GKST15] Rohit Gurjar, Arpita Korwar, Nitin Saxena, and Thomas Thierauf. Deterministic identity testing for sum of read-once oblivious arithmetic branching programs. In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, pages 323–346, 2015.

[GL99] Anna Galluccio and Martin Loebl. On the theory of Pfaffian orientations. I. perfect matchings and permanents. *Electr. J. Comb.*, 6, 1999.

[Gup15] Ankit Gupta. *Arithmetic Circuits: Lower Bounds, Derandomization and Reconstruction.* PhD thesis, Chennai Mathematical Institute, 2015.

[Hoa10] Thanh Minh Hoang. On the matching problem for special graph classes. In *IEEE Conference on Computational Complexity*, pages 139–150. IEEE Computer Society, 2010.

[HS80] J. Heintz and C. P. Schnorr. Testing polynomials which are easy to compute (extended abstract). In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pages 262–272, New York, NY, USA, 1980. ACM.

[HT73] John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973.

[IMZ12] Russell Impagliazzo, Raghu Meka, and David Zuckerman. Pseudorandomness from shrinkage. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 111–119, 2012.

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC, pages 356–364, New York, NY, USA, 1994. ACM.

[Jor69] Camille Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869.

[JQS10a] Maurice J. Jansen, Youming Qiao, and Jayalal Sarma. Deterministic black-box identity testing π-ordered algebraic branching programs. In *FSTTCS*, pages 296–307, 2010.

[JQS10b] Maurice J. Jansen, Youming Qiao, and Jayalal Sarma. Deterministic identity testing of read-once algebraic branching programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:84, 2010.

[Kas67] P W Kastelyn. Graph theory and crystal physics. *Graph Theory and Theoretical Physics*, pages 43–110, 1967.

[Khu88] S. Khuller. *Parallel Algorithms for K_5-minor Free Graphs.* Cornell University, Department of Computer Science, 1988.

[KI03] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *STOC*, pages 355–364, 2003.

[KMV08] Raghav Kulkarni, Meena Mahajan, and Kasturi R. Varadarajan. Some perfect matchings and perfect half-integral matchings in NC. *Chicago Journal of Theoretical Computer Science*, 2008(4), September 2008.

[KNP11] Michal Koucký, Prajakta Nimbhorkar, and Pavel Pudlák. Pseudorandom generators for group products: extended abstract. In *STOC*, pages 263–272, 2011.

[Koi12] Pascal Koiran. Arithmetic circuits: The chasm at depth four gets wider. *Theoretical Computer Science*, 448:56–65, 2012.

[Kor09] Arpita Korwar. Matching in planar graphs. Master's thesis, Indian Institute of Technology Kanpur, 2009.

[Kor15] Arpita Korwar. *PIT and lower bounds for sum of special ABPs.* PhD thesis, IIT Kanpur, 2015. (to be published).

[KR98] M. Karpiński and W. Rytter. *Fast Parallel Algorithms for Graph Matching Problems.* Oxford lecture series in mathematics and its applications. Clarendon Press, 1998.

[Kro82] Leopold Kronecker. *Grundzuge einer arithmetischen Theorie der algebraischen Grossen.* Berlin, G. Reimer, 1882.

[KS01] Adam Klivans and Daniel A. Spielman. Randomness efficient identity testing of multivariate polynomials. In *STOC*, pages 216–223, 2001.

[KS07] Neeraj Kayal and Nitin Saxena. Polynomial identity testing for depth 3 circuits. *Computational Complexity*, 16(2):115–138, 2007.

[KS09] Neeraj Kayal and Shubhangi Saraf. Blackbox polynomial identity testing for depth 3 circuits. In *FOCS*, pages 198–207, 2009.

[KS11] Zohar Shay Karnin and Amir Shpilka. Black box polynomial identity testing of generalized depth-3 arithmetic circuits with bounded top fan-in. *Combinatorica*, 31(3):333–364, 2011.

[KUW86] Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

[Lin92] Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 400–404, New York, NY, USA, 1992. ACM.

[LMR07] Nutan Limaye, Meena Mahajan, and B.V.Raghavendra Rao. Arithmetizing classes around NC^1 and L. In *STACS 2007*, volume 4393 of *Lecture Notes in Computer Science*, pages 477–488. Springer Berlin Heidelberg, 2007.

[Lov79] László Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574, 1979.

[LP86] L. Lovász and M.D. Plummer. *Matching Theory.* (North-Holland mathematics studies). Elsevier Science Ltd, 1986.

[MN95] Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM Journal on Computing*, 24:1002–1017, 1995.

[MV80] Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, SFCS '80, pages 17–27, Washington, DC, USA, 1980. IEEE Computer Society.

[MV97] Meena Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago J. Theor. Comput. Sci.*, 1997.

[MV00] Meena Mahajan and Kasturi R. Varadarajan. A new NC algorithm for finding a perfect matching in bipartite planar and small genus graphs (extended abstract). In *STOC'00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 351–357, New York, NY, USA, 2000. ACM.

[MVV87] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

[Nis90] N. Nisan. Pseudorandom generators for space-bounded computations. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 204–212, New York, NY, USA, 1990. ACM.

[Nis91] Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In *Proceedings of the 23rd ACM Symposium on Theory of Computing, ACM Press*, pages 410–418, 1991.

[NTS95] Noam Nisan and Amnon Ta-Shma. Symmetric logspace is closed under complement. In Frank Thomson Leighton and Allan Borodin, editors, *STOC*, pages 140–146. ACM, 1995.

[Pap94] Christos M. Papadimitriou. *Computational complexity.* Addison-Wesley, Reading, Massachusetts, 1994.

[RA00] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55:17:1–17:24, September 2008.

[RR99] Ran Raz and Omer Reingold. On recycling the randomness of states in space bounded computation. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 159–168, 1999.

[RS03] Neil Robertson and P.D Seymour. Graph minors. XVI. Excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 89(1):43 – 76, 2003.

[RS05] Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14(1):1–19, 2005.

[RY09] Ran Raz and Amir Yehudayoff. Lower bounds and separations for constant depth multilinear circuits. *Computational Complexity*, 18(2):171–207, 2009.

[Sax08] Nitin Saxena. Diagonal circuit identity testing and lower bounds. In *ICALP*, volume 5125 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 2008.

[Sax09] Nitin Saxena. Progress on polynomial identity testing. *Bulletin of the EATCS*, 99:49–79, 2009.

[Sax14] Nitin Saxena. Progress on polynomial identity testing- II. In *Perspectives in Computational Complexity*, volume 26 of *Progress in Computer Science and Applied Logic*, pages 131–146. Springer International Publishing, 2014.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980.

[SS11] Nitin Saxena and C. Seshadhri. An almost optimal rank bound for depth-3 identities. *SIAM J. Comput.*, 40(1):200–224, 2011.

[SS12] Nitin Saxena and C. Seshadhri. Blackbox identity testing for bounded top-fanin depth-3 circuits: The field doesn't matter. *SIAM J. Comput.*, 41(5):1285–1298, 2012.

[SS13] Nitin Saxena and C. Seshadhri. From Sylvester-Gallai configurations to rank bounds: Improved blackbox identity test for depth-3 circuits. *J. ACM*, 60(5):33, 2013.

[SSS09] Chandan Saha, Ramprasad Saptharishi, and Nitin Saxena. The power of depth 2 circuits over algebras. In *FSTTCS*, pages 371–382, 2009.

[Ste12] Thomas Steinke. Pseudorandomness for permutation branching programs without the group theory. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:83, 2012.

[STW14] Simon Straub, Thomas Thierauf, and Fabian Wagner. Counting the number of perfect matchings in K_5-free graphs. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 66–77, 2014.

[SV09] Amir Shpilka and Ilya Volkovich. Improved polynomial identity testing of read-once formulas. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, volume 5687 of LNCS*, pages 700–713, 2009.

[SW97] Petr Savický and Ingo Wegener. Efficient algorithms for the transformation between different types of binary decision diagrams. *Acta Informatica*, 34(4):245–256, 1997.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.

[Tav13] Sébastien Tavenas. Improved bounds for reduction to depth 4 and depth 3. In *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013. Proceedings*, pages 813–824, 2013.

[Tod91] Seinosuke Toda. Counting problems computationally equivalent to computing the determinant. *Technical Report CSIM*, 07, 1991.

[Tut47] W. T. Tutte. The Factorization of Linear Graphs. *Journal of the London Mathematical Society*, s1-22(2):107–111, 1947.

[TV12] Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. *Inf. Comput.*, 215:1–7, 2012.

[TW14] Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free and K_5-free graphs is in unambiguous logspace. *Chicago J. Theor. Comput. Sci.*, 2014.

[Val79] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Vaz89] Vijay V. Vazirani. NC algorithms for computing the number of perfect matchings in $K_{3,3}$-free graphs and related problems. *Information and Computing*, 80(2):152–164, 1989.

[vBV83] Burchard von Braunmühl and Rutger Verbeek. Input-driven languages are recognized in $\log n$ space. In *Foundations of Computation Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 40–51. Springer Berlin Heidelberg, 1983.

[Wag37] Klaus Wagner. Über eine Eigenschaft der ebenen Komplexe. *Math. Ann.*, 114, 1937.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, EUROSAM '79, pages 216–226, London, UK, UK, 1979. Springer-Verlag.

Index

3-bond, 93, 108

algebra, 18
arithmetic branching programs (ABP), 5, 24
arithmetic circuits, 2
articulation point, 86

basis isolating weight assignment, 34
basis isolation, 34
biconnected graphs, 86
bipartite graphs, 77
bivariate ROABP, 60
blackbox test, 3

circulation, 87
circulation of a face, 87
clique-sum, 91
coefficient of a monomial, $\text{coef}_P(\cdot)$, 17
coefficient space, 19
commutative ROABP, 7, 27, 66
component tree, 92
Count-PM, 14

Decision-PM, 11
degree, 18
depth-2 circuits, 4, 20
depth-3 circuits, 4, 22
diagonal circuits, 4, 22
dual graph, 88
duality trick, 22

formulas, 6

graph decomposition, 91

Hadamard algebra, 18
height of a node, 97
H-free graphs, 14, 84
hitting-set, 3, 20

individual degree, 18
isolating weight assignment, 79
Isolation lemma, 12, 80
isolation of a matching, 79

$K_{3,3}$, 14, 84
$K_{3,3}$-free graphs, 14, 84
K_5, 14, 84
K_5-free graphs, 14, 84
k-clique-sum, 91
Kronecker map, 21

Lagrange interpolation, 71
ℓ-concentration, 31
level of a node, 97
low-support concentration, 31

matching, 10, 76
matching polytope, 77
Matrix algebra, 18
multilinear circuit, 4

NC, 10, 75
NC^i, 75

ordered binary decision diagrams (OBDD), 48

partial derivative matrix, 60
perfect matching, 10, 76
perfect matching polytope, 77
planar graphs, 14, 77, 84
polynomial identity testing (PIT), 1
polynomial over an algebra, 19
polynomial over matrices, 18
projection of a cycle, 95
pseudorandom generators (PRG), 9, 58

quasi-polynomial time, 17

rank concentration, 31
read-once formula, 7

read-once oblivious ABP (ROABP), 6, 25
read-once ordered branching programs (ROBP), 9, 48
RNC, 10

Schwartz-Zippel lemma, 19
Search-PM, 11
separating pair, 92
separating set, 92
separating triplet, 92
set-multilinear circuits, 5, 23
shifting of variables, 32
skew-symmetric weight function, 87
sparse polynomials, 20
sparse-factor ROABP, 26
SPL, 76
s-sparse polynomial, 20
sum of products of univariates, 5, 22
support of a monomial, Supp$(\cdot)$, 17
support size, supp$(\cdot)$, 17

tree traversal, 85

variable order, 6, 25, 49
vertex-split, 106

whitebox test, 3
width of an ABP, 6, 25
width of an ROBP, 49
width-3 ABP, 6
working tree, 96, 104