On Approximate Polynomial Identity Testing and Real Root Finding

A study of approximating the rank of symbolic matrices and real roots of sparse polynomials

Gorav Jindal

Dissertation

zur Erlangung des Grades des
Doktors der Naturwissenschaften

an der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

Saarbrücken, Germany
2019

COLLOQUIUM INFORMATION

Date:	11.11.2019 Saarbrücken, Germany
Dean:	Prof. Dr. Sebastian Hack *Saarland University*
Chairman:	Prof. Dr. Raimund Seidel *Saarland University*
Reviewers:	Prof. Dr. Markus Bläser *Saarland University*
	Prof. Dr. Michael Sagraloff *Hochschule für angewandte Wissenschaften Landshut*
Scientific Assistant:	Dr. Balagopal Komarath *Saarland University*

Dedicated to My Parents and My Siblings

ACKNOWLEDGMENTS

Work in this thesis was carried out during the years 2014-2018 at the Max-Planck-Institut für Informatik and at the Department of Computer Science, Saarland University.

I owe my deepest gratitude to my advisor Prof. Markus Bläser. He always supported and guided me with his reliable expertise in algebraic complexity theory. He always listened to even the dumbest of my ideas and problems. He has been a pillar of support even outside the academics. Without his continuous encouragement and support, this thesis would not even have started.

I would also like to express my gratitude to Prof. Michael Sagraloff. His technical expertise and his infinite patience while answering my dumb queries, has helped me to learn a lot about the root computation of polynomials. The work done with him is an integral part of this thesis. I am also grateful to him for being a reviewer of this thesis.

I also received generous support from the Max-Planck-Institut für Informatik, especially from Prof. Kurt Mehlhorn. Thanks also to Prof. Chandan Saha for hosting me for an internship at the Indian Institute of Science. He has always been encouraging since the time he taught me here at Saarbrücken. I would also like to thank Prof. Parinya Chalermsook for his generous support for me at Aalto University.

Some special words of gratitude go to my friends who have always been a major source of support. Andi, Anurag, Eig, Harry, Laszlo, Karteek, Pavel, Shay have always been there for me to offer their unconditional support. I hope we keep crossing paths in future also. Special thanks to Andi for encouraging for me to get into cycling and always helping me to fix my almost always broken bike.

I sincerely apologize to all the people who directly or indirectly supported me but could not be mentioned here due to my inattention.

Finally, I would like to show my greatest appreciation to my family for their unwavering support. My parents and siblings have always made me feel loved and cherished. Without their unconditional support, I would not have the courage to embark on a research career.

ABSTRACT

In this thesis we study the following three topics, which share a connection through the (arithmetic) circuit complexity of polynomials.

1. Rank of symbolic matrices.
2. Computation of real roots of real sparse polynomials.
3. Complexity of symmetric polynomials.

We start with studying the commutative and non-commutative rank of symbolic matrices with linear forms as their entries. Here we show a deterministic polynomial time approximation scheme (PTAS) for computing the commutative rank. Prior to this work, deterministic polynomial time algorithms were known only for computing a $\frac{1}{2}$-approximation of the commutative rank. We give two distinct proofs that our algorithm is a PTAS. We also give a min-max characterization of commutative and non-commutative ranks.

Thereafter we direct our attention to computation of roots of uni-variate polynomial equations. It is known that solving a system of polynomial equations reduces to solving a uni-variate polynomial equation. We describe a polynomial time algorithm for (n, k, τ)-nomials which computes approximations of all the real roots (even though it may also compute approximations of some complex roots). Moreover, we also show that the roots of integer trinomials are *well-separated*.

Finally, we study the complexity of symmetric polynomials. It is known that symmetric Boolean functions are easy to compute. In contrast, we show that the assumption $\mathsf{VP} \neq \mathsf{VNP}$ implies that there exist hard symmetric polynomials. To prove this result, we use an algebraic analogue of the classical Newton iteration.

ZUSAMMENFASSUNG

In dieser Dissertation untersuchen wir die folgenden drei Themen, welche durch die (arithmetische) Schaltkreiskomplexität von Polynomen miteinander verbunden sind:

1. der Rang von symbolischen Matrizen,
2. die Berechnung von reellen Nullstellen von dünnbesetzten ("sparse") Polynomen mit reellen Koeffizienten,
3. die Komplexität von symmetrischen Polynomen.

Wir untersuchen zunächst den kommutativen und nicht-kommutativen Rang von Matrizen, deren Einträge aus Linearformen bestehen. Hier beweisen wir die Existenz eines deterministischem Polynomialzeit-Approximationsschemas (PTAS) für die Berechnung des kommutative Ranges. Zuvor waren polynomielle Algorithmen nur für die Berechnung einer $\frac{1}{2}$-Approximation des kommutativen Ranges bekannt. Wir geben zwei unterschiedliche Beweise für den Fakt, dass unser Algorithmus tatsächlich ein PTAS ist. Zusätzlich geben wir eine min-max Charakterisierung des kommutativen und nicht-kommutativen Ranges.

Anschließend lenken wir unsere Aufmerksamkeit auf die Berechnung von Nullstellen von univariaten polynomiellen Gleichungen. Es ist bekannt, dass das Lösen eines polynomiellem Gleichungssystems auf das Lösen eines univariaten Polynoms zurückgeführt werden kann. Wir geben einen Polynomialzeit-Algorithmus für (n, k, τ)-Nome, welcher Abschätzungen für alle reellen Nullstellen berechnet (in manchen Fallen auch Abschätzungen von komplexen Nullstellen). Zusätzlich beweisen wir, dass Nullstellen von ganzzahligen Trinomen stets weit voneinander entfernt sind.

Schließlich untersuchen wir die Komplexität von symmetrischen Polynomen. Es ist bereits bekannt, dass sich symmetrische Boolesche Funktionen leicht berechnen lassen. Im Gegensatz dazu zeigen wir, dass die Annahme $\mathsf{VP} \neq \mathsf{VNP}$ bedeutet, dass auch harte symmetrische Polynome existieren. Um dies zu beweisen benutzen wir ein algebraisches Analog zum klassischen Newton-Verfahren.

CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

1

INTRODUCTION

1.1 Motivation

To answer the Entscheidungsproblem posed by David Hilbert in 1928, Turing [TUR36] formalized the idea of an algorithm by defining the notion of a "Turing Machine". The famous Church-Turing thesis further states that Turing machine defined in [TUR36] captures the notion of any physical computation. Thus, it is reasonable to study the computation in context of Turing Machines. Turing machine as defined by Turing in [TUR36], only captures the notion of whether a function or number is computable or not. Turing did not ask, whether this computation is "efficient" or "inefficient"? This question of studying the "efficiency" of computation has lead to the field of computational complexity theory.

Gödel had pondered about the idea of efficient computation. In a letter [Har89] to von Neumann, he wrote:

I would like to allow myself to write you about a mathematical problem, of which your opinion would very much interest me: One can obviously easily construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n (length = number of symbols). Let $\psi(F,n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \psi(F,n)$. The question is how fast $\varphi(n)$ grows for an optimal machine. One can show that $\varphi(n) \geq K \cdot n$. If there really were a machine with $\varphi(n) \sim K.n$ (or even $\sim K \cdot n^2$), this would have consequences of the greatest importance. Namely, it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. After all, one would simply have to choose the natural number n so large that when the machine does not deliver a result, it makes no sense to think more about the problem. Now it seems to me, however, to be completely within the realm of possibility that $\varphi(n)$ grows that slowly. Since it seems that $\varphi(n) \geq K \cdot n$ is the only estimation which one can obtain by a generalization of the proof of the undecidability of the Entscheidungsproblem and

after all $\varphi(n) \sim K.n$ (or $\sim K \cdot n^2$) only means that the number of steps as opposed to trial and error can be reduced from N to $\log N$ (or $(\log N)^2$).

This letter illustrates that Gödel had already thought about whether there exists an algorithm to decide if a given mathematical statement has a reasonably sized proof. This intuition about whether proof finding can be automated has led to the "P vs NP" problem [Cooo0], which essentially asks whether finding proofs is as easy as verifying them?

The "P vs NP" problem has significantly contributed to development of computational complexity theory. Here the main goal is to classify computational problems in complexity classes. Goal of an algorithm designer is to find efficient algorithms for a given computational task. On the contrary, complexity theory is mainly concerned with trying to prove that there exists no efficient algorithm for a specified computational task. This boundary between algorithm design and complexity theory is very fine. Nevertheless, traditionally algorithm design and complexity theory have been in a race to prove the tractability or hardness of computational problems.

But these two seemingly competing goals have a lot in common. More specifically, it has been shown that finding deterministic algorithms for certain problems is equivalent to proving lower bounds [ACR98; Bab+93; IW97; ISW99; ISW00; NW94; SU01; STV01; Uma03; KI04]. For instance, the following theorem was proved in [IW97].

Theorem 1.1 (Theorem 2 in [IW97]). *If there is a language $f \in \mathsf{E}$ with $C(f_n) = 2^{\Omega(n)}$, then $\mathsf{BPP} = \mathsf{P}$.*

Here E is the complexity class of the languages which can be decided in time $O(2^{O(n)})$ by Turing machines. The complexity class BPP is essentially the set of languages which can be decided by a randomized polynomial time Turing machine and the class P is the set of languages which can be decided by a deterministic polynomial time Turing machine (see Chapter 2 for more precise definitions). This question $\mathsf{P} \stackrel{?}{=} \mathsf{BPP}$ is of fundamental importance in computational complexity and the philosophy of the nature of computation. $\mathsf{P} \stackrel{?}{=} \mathsf{BPP}$ is essentially asking whether randomness helps in computation at all? It is widely believed that $\mathsf{P} = \mathsf{BPP}$. Impagliazzo and Wigderson [IW97] prove that if we can prove strong enough circuit lower bounds on the complexity class E then we can derandomize BPP. This phenomenon is usually known as "Hardness vs Randomness". Theorem 1.1 implies that hardness leads to derandomization. We can also ask this question in the other direction, *i.e.*, whether derandomization implies hardness? Consider the following problem.

Problem 1.1 (Arithmetic Circuit identity testing, ACIT). *Given an arithmetic circuit C, decide if the polynomial computed by C is the zero polynomial?*

For a precise definition of arithmetic circuits, see Chapter 2. By using the Schwartz-Zippel Lemma (Lemma 3.1) of Chapter 3, one can show that Problem 1.1 (ACIT) is in the complexity class BPP, *i.e.*, can be solved in polynomial time by a randomized

polynomial time Turing machine. It is a big open question whether one can find deterministic polynomial time algorithms for ACIT. Now consider the following theorem from [KI04].

Theorem 1.2 (Theorem 18 in [KI04]). *If* ACIT *over* $\mathbb{Z}$ *can be solved in non-deterministic sub-exponential time then at least one of the following statement is true.*

1. NEXP *can not be solved by polynomial size Boolean circuits.*
2. *Over* $\mathbb{Q}$*, permanent can not be computed by polynomial-size arithmetic circuits.*

For a precise formulation of non-determinism, see Chapter 2. The complexity class NEXP is the set of languages which can be solved in non-deterministic $O(2^{\mathsf{poly}(n)})$ time. Thus Theorem 1.2 states that solving ACIT even in sub-exponential time (even with non-determinism) implies circuit lower bounds. Theorem 1.1 and Theorem 1.2 imply that derandomization and lower bounds are essentially two sides of the same coin. As we shall see in Chapter 2, that we can also study the complexity of polynomials instead of Boolean functions. The algebraic analogue of P vs NP problem is the VP vs VNP problem, also know as Valiant's Conjecture.

There is a surprising connection between the VP vs VNP problem and the real roots of sum of products of sparse polynomials. Consider a real uni-variate polynomial $f(x) \in \mathbb{R}[x]$ of the following form.

$$f(x) = \sum_{i=1}^{k} \prod_{j=1}^{m} f_{ij}(x). \tag{1.1}$$

Assume that $f_{ij}(x)$'s are real uni-variate polynomials having at most t monomials but can have arbitrarily high degree. Thus the degree of $f(x)$ is unbounded whereas $f(x)$ has at most $k \cdot t^m$ many monomials. By using the Descartes's rule of sign (see Chapter 5 and Theorem 5.3), we know that $f(x)$ can have at most $2k \cdot t^m$ many real roots. Now consider the following conjecture of Koiran [Koi11].

Conjecture 1.1 (Conjecture 3 in [Koi11], Real τ-conjecture). *The number of real roots of* f *in Equation (1.1), is bounded by a polynomial function of* kmt*.*

Koiran [Koi11] also proved that the real τ-conjecture (Conjecture 1.1) also implies lower bounds.

Theorem 1.3 ([Koi11]). *If real* τ*-conjecture is true then permanent is not in* VP^0*.*

Here VP^0 is the set of families of polynomials which can be computed by constant free polynomial size arithmetic circuits. In this thesis, we do not focus on proving the bounds on number of real roots of polynomials of the form Equation (1.1). Rather we focus on computing the real roots of sparse (polynomials having few monomials) uni-variate polynomials.

Motivated by these surprising connections, this thesis deals with the following three themes.

1. Polynomial identity testing.
2. Computing the real roots of real sparse polynomials.
3. Complexity of symmetric polynomials.

1.2 Contribution and Guide

In this section, we describe the structure of this thesis and contribution of various chapters. This thesis has three parts, described below in Subsections 1.2.1 to 1.2.3. The preliminaries Chapter 2 formalizes the necessary background of the themes encountered in this thesis.

1.2.1 Rank of matrix spaces

This part deals with computing the rank of matrix spaces and it is partially based on our contribution in [BJP18].

Chapter 3 first describes the classical Schwartz-Zippel Lemma[Sch80; Zip79], which is applied several times in this thesis. Thereafter, we formalize the classical polynomial identity testing problems. Chapter 3 then sets up the necessary background to define the notion of a matrix space. We also describe the notions of commutative and non-commutative ranks of a matrix space. To motivate the idea of commutative rank, several classical problems are demonstrated which reduce to computing the commutative rank of a matrix space. These problems are:

1. Maximum matching in bipartite graphs.
2. Maximum matching in general graphs using Tutte matrix
3. Identity testing of formulas and algebraic branching programs.

We show in Chapter 3, that computing the commutative rank of a matrix space (or equivalently of a symbolic matrix) is equivalent to the identity testing of algebraic branching programs. Thus computation of the commutative rank can be easily performed in randomized polynomial time but computing it deterministically remains elusive.

This chapter also describes several equivalent definitions of the non-commutative rank. We also prove the classical result of [FR04], which states that for any matrix space $\mathcal{B}$ the following inequality always holds.

$$\operatorname{crk}(\mathcal{B}) \leq \operatorname{ncrk}(\mathcal{B}) \leq 2 \cdot \operatorname{crk}(\mathcal{B}). \tag{1.2}$$

Chapter 3 concludes with a new max-min and min-max characterizations of commutative and non-commutative ranks respectively.

It was shown in [Gar+16] that $\mathrm{ncrk}(\mathcal{B})$ can be computed in deterministic polynomial time. In light of this result of [Gar+16], Equation (1.2) implies that a $\frac{1}{2}$-approximation of $\mathrm{crk}(\mathcal{B})$ can be computed in deterministic polynomial time. This motivates the question whether one can compute better approximations of the commutative rank deterministically. To answer this question, Chapter 4 describes a deterministic polynomial time algorithm for approximating the commutative rank of a given matrix space. This chapter describes our contribution published in [BJP18]. We introduce the ideas of Wong sequences in this chapter and a novel notion of Wong index. We show that the higher the Wong index of a given matrix, the more closely it approximates the commutative rank. This was the crucial contribution of [BJP18]. We also generalize this connection of Wong index and commutative rank, to non-commutative rank also. Moreover, Chapter 4 also describes an alternative approach to prove that the greedy algorithm of [BJP18] approximates the commutative rank. In this new approach, instead of looking at the Wong index, we look at the constant degree part of a suitable matrix polynomial. We hope that this new approach can be used to approximate the commutative rank in a more general setting as well.

1.2.2 Real roots of real sparse polynomials

The second part of this thesis deals with real roots of sparse polynomials. This part is based on our contribution in [JS17].

In Chapter 5, we first lay the foundation for studying the structure of roots of real sparse polynomials. It starts with a classical algebraic proof of the fundamental theorem of algebra. Then we formalize the notion of sparse polynomials. A simple proof for the Descartes's rule of signs is also presented. To describe our root computation algorithms, we need a generalization of Descartes's rule of signs. This generalization (described by using Obreshkoff regions) is introduced. Thereafter, we prove a root separation lower bound for the roots of integer trinomials. We also demonstrate a simple 4-nomial where such a lower bound is not possible.

Subsequently, Chapter 5 deals with computing the real roots of real sparse polynomials. We first explain the classical algorithm of Cucker, Koiran, and Smale [CKS99]. This algorithm computes all the integer roots of a sparse integer polynomial in polynomial time. This algorithm is used to motivate the idea of so called fractional derivatives. This notion of fractional derivatives is crucial to our algorithm which computes the real roots of real polynomials. Then we introduce the notions of weak coverings and (strong) coverings. These are the objects which describe the approximations of real roots of the given polynomial. We then describe an iterative algorithm to compute a weak covering of sparse polynomials. This weak covering is then converted to a (strong) covering by using the so called T_l-test based on Pellet's Theorem. The classical T_l-test is not efficient

if it is applied naively to sparse polynomials. Thus we propose a modified T_l-test which is efficient even for sparse polynomials. This allows us to compute a (strong) covering for sparse polynomials in polynomial time.

1.2.3 Complexity of symmetric polynomials

The last part studies the arithmetic complexity of symmetric polynomials.

It is known that symmetric Boolean functions are easy to compute. More precisely, every symmetric Boolean function can be computed using constant depth threshold circuits. These circuits can be captured by the complexity class TC^0. Then we consider the problem of checking whether a given Boolean function (resp. polynomial) is symmetric? We show that symmetry checking for Boolean functions is NP-hard (under Turing reductions), whereas symmetry checking for polynomials can be done in randomized polynomial time.

At last, we consider the arithmetic complexity of symmetric polynomials. In contrast to symmetric Boolean functions, we show that there exist symmetric polynomial families which have super polynomial arithmetic complexity (under the assumption that $\mathsf{VP} \neq \mathsf{VNP}$). This is shown using the classical Newton's Iteration in the algebraic setting.

2

PRELIMINARIES

The main theme of this thesis is computation in algebraic and numerical models. In the classical Boolean model, one studies the complexity of Boolean functions. This complexity is usually studied in terms of the size of the smallest Boolean circuit computing the given Boolean function. In this chapter, we motivate the study of algebraic models of computation. Moreover we define the notion of algebraic circuits and related concepts. For a more comprehensive introduction, we refer the reader to [Büoo; SY+10; AB09]. Below are some of the notations which are used frequently.

2.1 Notation

1. For a natural number $n \in \mathbb{N}$, we use the notation $[n]$ to denote the set $\{1, 2, \ldots, n\}$, also $[[n]]$ is used to denote the set $[n] \cup \{0\} = \{0, 1, 2, \ldots, n\}$.
2. If $\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_n} \in \mathbb{F}^m$ are n column vectors then we use the notation $[\mathbf{c_1}; \mathbf{c_2}; \ldots; \mathbf{c_n}]$ to denote the $m \times n$ matrix whose i^{th} column is $\mathbf{c_i}$.
3. $f^{[d]}$ is used to denote the degree d homogeneous component of a polynomial f.

2.2 Boolean and Algebraic Circuits

Definition 2.1 (Boolean Circuit)**.** A Boolean circuit C is a finite directed acyclic graph. Each vertex is either a $\{\neg, \wedge, \vee\}$-gate or one of the inputs, and there is exactly one node which is labeled as the output.

If the given Boolean circuit C has n inputs $x_1, x_2, \ldots, x_n$ then it naturally computes a function $C_f : \{0,1\}^n \to \{0,1\}$ in the following way.

- Input gates x_i compute the Boolean function x_i.
- $\neg$-gate g computes the Boolean function $\neg(h)$ where h is the Boolean function computed by the child of g.

- $\wedge$-gate g computes the Boolean function $(h_1 \wedge h_2)$ where h_1 and h_2 are the Boolean functions computed by the children g_1, g_2 of g.
- $\vee$-gate g computes the Boolean function $(h_1 \vee h_2)$ where h_1 and h_2 are the Boolean functions computed by the children g_1, g_2 of g.

We say that the function computed by the output gate of C is the function computed by C, denoted by C_f. Note that we assume $\{\wedge, \vee\}$-gates have in degree two. The *depth* of a Boolean circuit is defined as the length of longest path from an input node to the output node. The *size* of the circuit is defined as the number of gates in it. For a given Boolean function $f : \{0,1\}^n \to \{0,1\}$, one studies the smallest (with respect to the size) Boolean circuit which computes f. To this end, we define the following complexity measure.

Definition 2.2 (Boolean Circuit Complexity)**.** The circuit complexity $C(f)$ of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is the size of the smallest circuit computing f.

Proving lower bounds on the circuit complexity of Boolean functions is the main goal of computational complexity. Unfortunately, this task has remained elusive in the case of general Boolean circuits defined above. There has been significant progress in proving lower bounds in restricted circuit models. We refer the reader to [AB09; Vol13; Juk12] for a more comprehensive introduction to circuit complexity. Now it is easy to observe that the following equalities for Boolean functions h, h_1, h_2 hold:

$$\begin{aligned} \neg h &= 1 - h \\ h_1 \wedge h_2 &= h_1 \times h_2 \\ h_1 \vee h_2 &= h_1 + h_2 - h_1 \times h_2 \end{aligned}$$

This suggests that Boolean gates can be simulated using the algebraic operations $\{+, -, \times\}$. This in turn implies that Boolean functions can be computed using $\{+, -, \times\}$ gates instead of $\{\neg, \wedge, \vee\}$ gates. Thus we define the following idea of arithmetic circuits.

Definition 2.3 (Arithmetic Circuit)**.** An arithmetic circuit C is a finite directed acyclic graph. Each vertex is one of the following:

- An input gate labeled by some variable x_i with in degree zero.
- A constant gate with in degree zero, labeled by some constant $c \in \mathbb{F}$. Here $\mathbb{F}$ is the underlying field.
- A $\{+, -, \times\}$-gate with in degree two.
- An output gate with out degree zero, we assume there is exactly one output gate.

Figure 2.1: Example of an arithmetic circuit

If the given arithmetic circuit C has n inputs $x_1, x_2, \ldots, x_n$ then it naturally computes a polynomial $C_p \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ in the following way.

- Variable input gates x_i compute the polynomial x_i.
- Constant input gates c compute the constant polynomial c.
- For $\circ \in \{+, -, \times\}$, a $\circ$-gate g computes the polynomial $(h_1 \circ h_2)$ where h_1 and h_2 are the polynomials by the children g_1, g_2 of g.

We say that the polynomial computed by the output gate of C is the polynomial computed by C, denoted by C_p. Usually, the output gate is also a $\{+, -, \times\}$-gate because otherwise C computes a trivial polynomial.

Remark 2.1. In Definition 2.3, we assumed that there is only one output gate. In a more general setting, one can consider arithmetic circuits which have multiple outputs and thus compute a set of polynomials. In the definition of arithmetic circuits, "$-$" gates are omitted sometimes in the literature. This is because $h_1 - h_2 = h_1 + (-1)h_2$ and thus "$-$" gates can be simulated by "$+$" gates.

The *size* of an arithmetic circuit is defined as the number of gates in it. For a polynomial $p \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, one studies the smallest arithmetic circuit computing p.

The polynomial computed by the arithmetic circuit in Figure 2.1 is $10x_3(x_1 + x_2) + x_1 + x_2 + x_4$.

Definition 2.4 (Arithmetic Circuit Complexity)**.** For a polynomial $p \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, the (arithmetic) circuit complexity $L(p)$ of p is defined as the size of smallest arithmetic circuit computing p, that is

$$L(p) \stackrel{\text{def}}{=\!=} \min\{s \mid \exists \text{ size } s \text{ arithmetic circuit computing } p\}.$$

We can also define the arithmetic complexity of a set $S \subseteq \mathbb{F}[x_1, x_2, \ldots, x_n]$ of polynomials. So $L(S)$ for a set $S \subseteq \mathbb{F}[x_1, x_2, \ldots, x_n]$ of polynomials is defined as the minimum size of any arithmetic circuit whose outputs compute all the elements of S.

Suppose we have a circuit C computing a polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$. It might be the case that f is not homogeneous. For some applications, it might be better to work with homogeneous polynomials. So we want to know if there exist "small" circuits also for the homogeneous components of f. The following Lemma 2.1 proves that the homogeneous components of f also have "small" arithmetic circuits.

Lemma 2.1 ([SY+10; Str73]). *For all polynomials f and all $d \in \mathbb{N}$, we have $L(f^{[d]}) \leq O(d^2 \cdot L(f))$.*

Proof. Let C be a circuit of size $L(f)$ computing f. We create $d+1$ copies of each arithmetic gate in C, *i.e.*, each$\{+,-,\times\}$-gate G has $d+1$ copies $G_0, G_1, \ldots, G_d$. If the gate G computes the polynomial g then G_i computes the polynomial $g^{[i]}$. This can be trivially done for input and constant gates. Suppose $G = G_1 + G_2$ is a "+" gate and g_1, g_2 are the polynomials computed by gates G_1 and G_2 respectively. Now we know that $g^{[i]} = g_1^{[i]} + g_2^{[i]}$ for all $i \in [[d]]$. A similar statement is true for "$-$" gates also. If $G = G_1 \times G_2$ is a "$\times$" gate then we have the following equality.

$$g^{[i]} = \sum_{j=0}^{i} g_1^{[j]} \cdot g_2^{[i-j]} \tag{2.1}$$

Suppose we already have the gates for $g_1^{[j]}, g_2^{[j]}$ for all $j \in [[d]]$. Then one $g^{[i]}$ in Equation (2.1) can be computed using $2(i+1)$ additional gates. Thus the gates $G_0, G_1, \ldots, G_d$ can be constructed using $\sum_{k=0}^{d} 2(k+1) = O(d^2)$ gates. Hence every gate in C corresponds to at most $O(d^2)$ new gates. Thus $L(f^{[d]}) \leq O(d^2 \cdot L(f))$. □

Remark 2.2. Note that the circuit constructed in the proof of Lemma 2.1 computes all the homogeneous components $f^{[0]}, f^{[1]}, \ldots, f^{[d]}$ instead of just $f^{[d]}$.

2.3 Complexity classes

- A function $q : \mathbb{N} \to \mathbb{N}$ is called polynomially bounded or simply p-bounded if there exists a polynomial $f \in \mathbb{Z}[x]$ such that for all $n \in \mathbb{N} : q(n) \leq f(n)$.

Definition 2.5 (Language)**.** A language L is just a subset of $\{0,1\}^*$, that is, a set of binary sequences of any length. For a language $L \subseteq \{0,1\}^*$, there is a corresponding total function L_f which *computes* L, that is, $L_f : \{0,1\}^* \to \{0,1\}$ and $L_f(x) = 1$ iff $x \in L$. We use the symbol L_n to denote the function L_f restricted on $\{0,1\}^n$.

Remark 2.3. In Definition 2.5, one can also define the languages over an arbitrary alphabet Σ. But the binary alphabet $\Sigma = \{0,1\}$ is the most commonly used alphabet.

Definition 2.6 (Complexity class)**.** A complexity class $\mathcal{C}$ is a set of languages.

Most complexity classes are defined by how much resources they need on some abstract machine (e.g Turing machines). We define below some well known complexity classes. For a more complete introduction to Turing machines and complexity classes, we refer the reader again to [ABo09]. For a complexity class $\mathcal{C}$, the complement complexity class co$\mathcal{C}$ of $\mathcal{C}$ is defined as

$$\text{co}\mathcal{C} \stackrel{\text{def}}{=\!=} \{\{0,1\}^* \setminus L \mid L \in \mathcal{C}\}.$$

2.3.1 Classes P, NP and completeness

We define the complexity classes using Boolean circuits. To this end, we need the notion of polynomial-time uniform family of Boolean circuits.

Definition 2.7 (Polynomial-time uniform family)**.** A family of Boolean circuits $\{C_n : n \in \mathbb{N}\}$ is said to be *polynomial-time uniform* if there exists a polynomial time deterministic Turing machine M, such that for all $n \in \mathbb{N}$, M outputs a description of C_n on input 1^n.

In Definition 2.7, for the description of C_n, any *reasonable* encoding of Boolean circuits can be used. Also, polynomial time deterministic Turing machine M outputting C_n implies that the size of C_n is p-bounded function of n.

Definition 2.8 (Complexity class P)**.** A language L is in the complexity class P if and only if there exists a polynomial-time uniform family of Boolean circuits $\{C_n : n \in \mathbb{N}\}$, such that C_n computes the function L_n for all $n \in \mathbb{N}$.

So the complexity class P is the set of decision problems (interchangeably used with languages) which can be decided by polynomial size circuits or in polynomial time by deterministic Turing machines. To motivate the definition of NP, consider the following decision problem CSAT.

Definition 2.9 (Language CSAT)**.** The language CSAT is defined as the set of (encoding of) of circuits which can be satisfied, that is,

$$\text{CSAT} \stackrel{\text{def}}{=\!=} \{\text{Circuit } C(x_1, x_2, \ldots, x_n) \mid \exists (a_1, a_2, \ldots, a_n) \in \{0,1\}^n : C(a_1, a_2, \ldots, a_n) = 1\}.$$

Here CSAT can be seen as a subset of $\{0,1\}^*$ by encoding the Boolean circuits using binary strings in a reasonable way. Given a (encoding of a) circuit $C(x_1, x_2, \ldots, x_n)$, how to check if it is satisfiable? At first it seems that one needs to evaluate C on all the 2^n assignment $(a_1, a_2, \ldots, a_n) \in \{0,1\}^n$ and then check if C evaluates to 1 on one of these 2^n assignments. In fact, essentially this is the best algorithm known for CSAT. The conjectures ETH and SETH [CIP09; IP01] state that this is the best one can do even in restricted models like 3-SAT and k-SAT. But now observe that if someone gives us an assignment $(a_1, a_2, \ldots, a_n) \in \{0,1\}^n$ which satisfies C ($C(a_1, a_2, \ldots, a_n) = 1$) then we can check the condition $C(a_1, a_2, \ldots, a_n) = 1$ efficiently, by evaluating C on $(a_1, a_2, \ldots, a_n)$. Such a satisfying assignment $(a_1, a_2, \ldots, a_n) \in \{0,1\}^n$ is called a "witness". The class of decision problems where such "short" witnesses exist and their validity can be verified in polynomial time, is called the complexity class NP. We formalize this definition below.

Definition 2.10 (Complexity class NP)**.** A language L is in the complexity class NP if and only if the following conditions are true.

- There exists a polynomial-time uniform family of Boolean circuits $\{C_n : n \in \mathbb{N}\}$.

- There exists a p-bounded function q such that
  - For all $x \in L$, there exists a "witness" $w \in \{0,1\}^{q(|x|)}$ such that:

$$C_{|x|+q(|x|)}(x, w) = 1.$$

  - For all $x \notin L$, no such "witness" w exists.

Usually, the class NP is defined as the set of decision problems solvable in polynomial time by a non-deterministic Turing machine. But it is an easy exercise to show that Definition 2.10 is an alternate definition of NP. To formalize the idea of hardness of a problem, we define the notion of a reduction.

Definition 2.11 (Many-one reduction)**.** Let R be some set of functions $\{0,1\}^* \to \{0,1\}^*$. A language L' is called R many one reducible to another language L if there is some function $f \in R$ (the reduction) such that for all $x \in \{0,1\}^*$,

$$x \in L' \iff f(x) \in L.$$

We use the notation $L' \leq_R L$ to denote that L' is R many one reducible to L.

In Definition 2.11, if R is the set of polynomial time computable functions then we say that L' polynomial time many one reduces to L, we denote this by $L' \leq_{\mathrm{P}} L$. An *oracle Turing machine* M with oracle access to a language L is denoted by M^L.

Definition 2.12 (Turing reduction)**.** Let R be a Turing machine. A language L' is called R Turing reducible to another language L if L' is the language decided by the oracle Turing machine R^L. We use the notation $L' \leq_R^{\mathrm{T}} L$ to denote that L' is R Turing reducible to L.

In Definition 2.12, if R is a polynomial time Turing machine then we say that L' polynomial time Turing reduces to L, we denote this by $L' \leq_{\mathrm{P}}^{\mathrm{T}} L$. We usually define the notion of hardness by using polynomial time many one reductions.

Definition 2.13 (Hardness and Completeness)**.** For a complexity complexity classic $\mathcal{C}$, a language L is said to be $\mathcal{C}$-hard if $L' \leq_{\mathrm{P}} L$ for all $L' \in \mathcal{C}$. L is said to be $\mathcal{C}$-complete if L is $\mathcal{C}$-hard and $L \in \mathcal{C}$.

Theorem 2.1 (Theorem 2.21 in [Golo8])**.** *The language* CSAT *is* NP*-complete.*

2.3.2 Low depth circuits

Proving super-polynomial lower bounds has proved to be a hard task. So it is natural to study the restricted classes of circuit families on which one can hopefully prove some lower bounds. To this end, we define the complexity classes NC, AC and TC. First we define the notion of a threshold gate.

Definition 2.14 (Threshold gate)**.** A threshold gate takes m inputs $x_1, x_2, \ldots, x_m$ and computes the following Boolean function $T_k : \{0,1\}^m \to \{0,1\}$:

$$T_k(x_1, x_2, \ldots, x_m) = \begin{cases} 1 & \sum_{i=1}^{m} x_i \geq k \\ 0 & \text{otherwise} \end{cases}$$

A majority gate $\text{MAJ}_m(x_1, x_2, \ldots, x_m)$ is defined as $T_{\lceil \frac{m}{2} \rceil}(x_1, x_2, \ldots, x_m)$.

Definition 2.15 (Complexity class NC^i)**.** A language L is in NC^i if there exists a polynomial-time uniform family of Boolean circuits $\{C_n : n \in \mathbb{N}\}$ such that C_n computes the function L_n for all $n \in \mathbb{N}$, the depth of C_n is bounded by $O(\log^i n)$ and the size of C_n is bounded by $\mathsf{poly}(n)$. Here the gates in C_n are of fan-in at most two.

Definition 2.16 (Complexity class AC^i)**.** A language L is in AC^i if there exists a polynomial-time uniform family of Boolean circuits $\{C_n : n \in \mathbb{N}\}$ such that C_n computes the function L_n for all $n \in \mathbb{N}$, the depth of C_n is bounded by $O(\log^i n)$ and the size of C_n is bounded by $\mathsf{poly}(n)$. Here the AND and OR gates in C_n have unlimited fan-in.

Definition 2.17 (Complexity class TC^i)**.** A language L is in TC^i if there exists a polynomial-time uniform family of Boolean circuits $\{C_n : n \in \mathbb{N}\}$ such that C_n computes the function L_n for all $n \in \mathbb{N}$, the depth of C_n is bounded by $O(\log^i n)$ and the size of C_n is bounded by $\mathsf{poly}(n)$. Here the AND and OR gates in C_n have unlimited fan-in. Also, C_n is allowed to have unlimited fan-in threshold gates.

Note that unlimited fan-in AND and OR gates are also threshold gates. Thus all the gates in a TC^i can be assumed to be threshold gates. By the definitions above, it is easy to see the following containment.

$$\forall i : \mathsf{NC}^i \subseteq \mathsf{AC}^i \subseteq \mathsf{TC}^i.$$

We shall see in Chapter 6 that the languages defined by symmetric Boolean functions are in TC^0.

Remark 2.4. Note that any threshold gate $T_k(x_1, x_2, \ldots, x_m)$ can be simulated by a majority gate as below:

$$T_k(x_1, x_2, \ldots, x_m) = \begin{cases} \text{MAJ}_{2k}(x_1, x_2, \ldots, x_m, 0, 0, \ldots, 0) & \text{If } m < 2k \\ \text{MAJ}_{2k}(x_1, x_2, \ldots, x_m, 1, 1, \ldots, 1) & \text{otherwise} \end{cases}$$

And therefore all the threshold gates in Definition 2.17 can be assumed to majority gates.

Although not relevant to the discussion here, Theorem 2.2 shows that proving lower bounds on restricted circuit classes can be easier. To this end, we define the following language PARITY.

$$\text{PARITY} \stackrel{\text{def}}{=\!=} \{x \in \{0,1\}^* \mid \text{ number of } 1\text{'s in } x \text{ is odd}\}.$$

Theorem 2.2 ([Smo93]). PARITY $\notin \mathsf{AC}^0$.

The complexity classes NC, AC and TC are defined as:

$$\mathsf{NC} \stackrel{\text{def}}{=\!=} \bigcup_{i \in \mathbb{N}} \mathsf{NC}^i$$
$$\mathsf{AC} \stackrel{\text{def}}{=\!=} \bigcup_{i \in \mathbb{N}} \mathsf{AC}^i$$
$$\mathsf{TC} \stackrel{\text{def}}{=\!=} \bigcup_{i \in \mathbb{N}} \mathsf{TC}^i$$

2.3.3 Randomized Complexity Classes

A randomized Turing machine is a Turing machine which can flip a fair coin in every step. With probability $\frac{1}{2}$, the outcome is 1 and otherwise it is 0.

Definition 2.18 (Complexity class RP). A language L is in RP if there exists a randomized Turing machine M with polynomial running time such that:

1. for all $x \in L$, M accepts x with probability at least $\frac{1}{2}$.
2. for all $x \notin L$, M rejects x with probability 1.

Turing machines in Definition 2.18 are said to have one-sided error. One can also define complexity classes with two-sided error. For instance:

Definition 2.19 (Complexity class BPP). A language L is in BPP if there exists a randomized Turing machine M with polynomial running time such that:

1. for all $x \in L$, M accepts x with probability at least $\frac{2}{3}$.
2. for all $x \notin L$, M rejects x with probability at least $\frac{2}{3}$.

2.4 Formulas and Algebraic Branching Programs

We have described above the model of algebraic circuits which compute polynomials. In this section, we describe the ideas of two other well studied algebraic models of computation.

Figure 2.2: Example of an algebraic branching program

Definition 2.20 (Arithmetic Formula)**.** An arithmetic circuit is called a formula if the underlying acyclic graph in Definition 2.3 is a tree.

It is obvious that arithmetic circuits are at least as *powerful* as arithmetic formulas because an arithmetic formula is trivially an arithmetic circuit. Arithmetic branching programs are another well studied algebraic model of computation.

Definition 2.21 (Algebraic Branching Program (ABP) [Nis91])**.** An Algebraic Branching Program (ABP) in variables $x_1, x_2, \ldots, x_n$ over the field $\mathbb{F}$ is a directed acyclic graph with the following properties.

1. There is a distinguished vertex s of in-degree zero (the source).
2. There is a distinguished vertex t of out-degree zero (the sink).
3. Each edge e is labeled with a polynomial f_e in the input variables $x_1, x_2, \ldots, x_n$.

The size of an ABP is defined as the number of vertices in the ABP. In the Definition 2.21, we have not imposed any restrictions on the edges. But it can be shown that with a polynomial blowup in the the size, ABPs can be assumed to layered. This means that that the vertices of the underlying graph are partitioned into layers $0, 1, \ldots, T$. In this case, edges in the graph are only allowed to go from layer $k-1$ to layer k , for $k \in [T]$. Thus the source vertex s is is the only vertex at layer 0 and the sink vertex t is the only vertex at layer T. So we always assumed ABPs to be layered.

The width of any ABP is the maximum number of nodes in any layer. The degree of an ABP is defined to be the maximal degree of the polynomial edge labels. We can define the polynomial computed by an ABP in the following way.

- Polynomial f_P computed by each $s \rightsquigarrow t$ path P is the product of the labels of the edges on P, *i.e.*, $f_P = \prod_{e \in P} f_e$.
- ABP A computes the polynomial f_A which is the sum of all the polynomials computed by all the $s \rightsquigarrow t$ paths, that is:

$$f_A = \sum_{P \text{ is an } s \rightsquigarrow t \text{ path}} f_P.$$

For example, the polynomial computed by the path $s \to a_1 \to b_2 \to t$ in Figure 2.2 is $(2x_1 + 5x_2 + 7) \cdot (27) \cdot (13x_1 + 17x_2)$. The polynomial computed by the ABP of Figure 2.2 is sum of all such polynomials. In Definition 2.21 of ABPs, we allowed the edge labels to be arbitrary polynomials. But this can allow even very short ABPs to compute hard

polynomials. So to define the idea of ABP complexity, we only allow linear polynomials as edge labels, as in Figure 2.2. Thus from now on, we always assume edge labels to be linear polynomials, that is, polynomials of degree at most 1.

Similar to the notion of arithmetic complexity, we can define the formula and ABP complexity.

Definition 2.22 (Arithmetic Formula Complexity)**.** For a polynomial $p \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, the (arithmetic) formula complexity $L_e(p)$ of p is defined as the size of smallest arithmetic formula computing p, that is

$$L_e(p) \stackrel{\text{def}}{=\!=} \min\{s \mid \exists \text{ size } s \text{ arithmetic formula computing } p\}.$$

In the notation L_e, e stands for *expression* which is an equivalent term for formulas. As hinted above, it is obvious that $\forall f \in \mathbb{F}[x_1, x_2, \ldots, x_n] : L(f) \leq L_e(f)$.

Definition 2.23 (ABP Complexity)**.** For a polynomial $p \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, the ABP complexity $L_a(p)$ of p is defined as the size of smallest ABP computing p, that is

$$L_a(p) \stackrel{\text{def}}{=\!=} \min\{s \mid \exists \text{ size } s \text{ ABP computing } p\}.$$

2.5 Algebraic Complexity classes

Analogous to the idea of classical complexity classes, we can also define the algebraic complexity classes. Note that we have defined the notion of arithmetic complexity in a non-uniform manner. This means that we do not require that the description of an arithmetic circuit computing the polynomial should be the output of some Turing Machine. And therefore we have to define algebraic complexity classes using polynomial families. We refer the reader to [Büo0; Mah14] for a more comprehensive introduction to algebraic complexity classes.

Definition 2.24 (p-family)**.** A family (or a sequence) $(f_n)_{n\in\mathbb{N}}$ of (multivariate) polynomials over the field $\mathbb{F}$ is said to be a p-family iff the number of variables as well as the degree of f_n are p-bounded functions of n.

Now we define the notion of *efficient* polynomial families.

Definition 2.25 (p-computable)**.** A p-family $(f_n)_{n\in\mathbb{N}}$ is called p-computable iff the arithmetic complexity $L(f_n)$ is a p-bounded function of n.

p-computable polynomial families define the algebraic analogue of the P, called VP.

Definition 2.26 (Class VP)**.** The (algebraic complexity) class VP is the set of all p-computable polynomial families.

Definition 2.27 (Class VP_e)**.** The (algebraic complexity) class VP_e is the set of all p-families $(f_n)_{n\in\mathbb{N}}$ such that $L_e(f_n)$ is a p-bounded function of n.

Definition 2.28 (Class VBP)**.** The (algebraic complexity) class VBP is the set of all p-families $(f_n)_{n\in\mathbb{N}}$ such that $L_a(f_n)$ is a p-bounded function of n.

A non-deterministic counterpart VNP of VP can be defined as follows.

Definition 2.29 (Class VNP)**.** A p-family $(f_n)_{n\in\mathbb{N}}$ is said to be in the (algebraic complexity) class VNP if there exists a polynomial family $(g_n)_{n\in\mathbb{N}} \in \mathsf{VP}$ with $g_n \in \mathbb{F}[x_1, x_2, \ldots, x_{q(n)}]$ such that:

$$f_n(x_1, x_2, \ldots, x_{p(n)}) = \sum_{e\in\{0,1\}^{q(n)-p(n)}} g_n(x_1, x_2, \ldots, x_{p(n)}, e_1, e_2, \ldots, e_{q(n)-p(n)}).$$

Analogous to Definition 2.29 of VNP, we can also define the class VNP_e where the polynomial family $(g_n)_{n\in\mathbb{N}}$ is required to be in class VP_e instead of VP. Surprisingly, it is known that $\mathsf{VNP}_e = \mathsf{VNP}$.

Theorem 2.3 ([Büoo])**.** *Over all fields,* $\mathsf{VNP}_e = \mathsf{VNP}$.

From the perspective of computational power, we also know that ABPs lie in between formulas and circuits.

Theorem 2.4 ([Mah14])**.** *Over all fields,* $\mathsf{VP}_e \subseteq \mathsf{VBP} \subseteq \mathsf{VP}$.

2.6 Completeness and hard polynomials

Similar to the Boolean case, there is an algebraic notion of reduction also, called p-projections.

Definition 2.30 (Projection)**.** A polynomial $f(x_1, x_2, \ldots, x_n) \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ is said to be a projection of a polynomial $g(y_1, y_2, \ldots, y_m) \in \mathbb{F}[y_1, y_2, \ldots, y_m]$, if there exists a map $\alpha : \{y_1, y_2, \ldots, y_m\} \to \{x_1, x_2, \ldots, x_n\} \cup \mathbb{F}$ such that $f = g$ under the substitution map α. We write $f \leq g$ to denote that f is a projection of g.

Definition 2.31 (p-projection)**.** A p-family $(f_n)_{n\in\mathbb{N}}$ is said to be a p-projection of a p-family $(g_n)_{n\in\mathbb{N}}$ if there is a p-bounded function $\beta : \mathbb{N} \to \mathbb{N}$ and $n_0 \in \mathbb{N}$ such that :

$$\forall n \geq n_0 : f_n \leq g_{\beta(n)}.$$

We denote $(f_n)_{n\in\mathbb{N}}$ being a p-projection of $(g_n)_{n\in\mathbb{N}}$ by $f \leq_p g$.

Now the idea of completeness and hardness can be defined as in the Boolean case.

Definition 2.32 (Hardness and Completeness)**.** For an algebraic complexity classic $\mathcal{C}$, a p-family $f = (f_n)_{n\in\mathbb{N}}$ is said to be $\mathcal{C}$-hard if $g \leq_p f$ for all $g \in \mathcal{C}$, f is called $\mathcal{C}$-complete if f is $\mathcal{C}$-hard and $f \in \mathcal{C}$.

It is known that the well known determinant polynomial family $(\det_n)$ is VBP-complete. Recall that $\det_n$ is defined as:

$$\det_n \stackrel{\text{def}}{=\!=} \sum_{\pi\in\mathfrak{S}_n} \operatorname{sgn}(\pi) \prod_{i=1}^{n} x_{i,\pi(i)}.$$

The following Theorem 2.5 shows that $(\det_n)$ is VP_e-hard.

Theorem 2.5 ([Val79])**.** *Let $f \in \mathbb{F}[x_1, x_2, \ldots, x_m]$ be a polynomial computed by an arithmetic formula of size s then f is a projection of $\det_{s+2}$. Additionally, this projection can be computed in deterministic $\mathsf{poly}(m, s)$ time.*

In fact, we know that ABPs are projections of small size determinants and vice-versa.

Theorem 2.6 ([Vin91; Tod91; MV97].)**.** *Let $f \in \mathbb{F}[x_1, x_2, \ldots, x_m]$ be a polynomial computed by an ABP of size s then f is a projection of $\det_s$. Additionally, this projection can be computed in deterministic $\mathsf{poly}(m, s)$ time. The converse is also true, i.e., $\det_s$ can be computed by a $\mathsf{poly}(s)$ size ABP.*

Corollary 2.1. *$(\det_n)$ is VP_e-hard and VBP-complete.*

By using Theorem 2.4, we conclude that $(\det_n) \in \mathsf{VP}$. It is not known whether $(\det_n)$ is VP-hard, although $(\det_n)$ is known to be VP-hard under so called qp-projections [Büoo]. We refer the curious reader to [Dur+14] for a discussion on VP-complete polynomial families. Now we define the well known VNP-complete polynomial family called the permanent.

$$\operatorname{per}_n \stackrel{\text{def}}{=\!=} \sum_{\pi\in\mathfrak{S}_n} \prod_{i=1}^{n} x_{i,\pi(i)}.$$

Theorem 2.7 ([Val79; Büoo])**.** *Over the fields $\mathbb{F}$ with $\operatorname{char}(\mathbb{F}) \neq 2$, p-family (per_n) is VNP-complete.*

The holy grail of algebraic complexity theory is to show that VP $\neq$ VNP. For this it is enough to show that $(\operatorname{per}_n) \notin \mathsf{VP}$ over fields $\mathbb{F}$ with $\operatorname{char}(\mathbb{F}) \neq 2$. Note that per_n and $\det_n$ are the same polynomials if $\operatorname{char}(\mathbb{F}) = 2$. Thus if $\operatorname{char}(\mathbb{F}) = 2$ then $(\operatorname{per}_n) \in \mathsf{VP}$. Hence (per_n) is unlikely to be VNP-complete over fields of characteristic two.

2.7 Definitions and Facts in Linear Algebra

Definition 2.33 (Null-space)**.** For an $m \times n$ matrix $A \in \mathbb{F}^{m\times n}$, the null-space $\mathrm{Ker}(A)$ of A is a subspace of $\mathbb{F}^n$ defined as below:

$$\mathrm{Ker}(A) = \{v \in \mathbb{F}^n \mid Av = 0\}.$$

Definition 2.34 (Image)**.** For an $m \times n$ matrix $A \in \mathbb{F}^{m\times n}$, the image $\mathrm{Im}(A)$ of A is a subspace of $\mathbb{F}^m$ defined as below:

$$\mathrm{Im}(A) = \{Av \mid v \in \mathbb{F}^n\}.$$

Remark 2.5. The rank of an $m \times n$ matrix $A \in \mathbb{F}^{m\times n}$ can be defined as $\dim(\mathrm{Im}(A))$.

Theorem 2.8 (Rank–nullity theorem, page 199 of [Meyoo])**.** *If A is an $m \times n$ matrix (with m rows and n columns) over some field, then we have:*

$$\mathrm{rank}(A) + \dim(\mathrm{Ker}(A)) = n.$$

Part I

POLYNOMIAL IDENTITY TESTING

3

SYMBOLIC MATRICES AND MATRIX SPACES

This chapter deals with checking the singularity of symbolic matrices. This singularity problem can be phrased in terms of the rank of corresponding matrix spaces. Here we introduce the definitions of symbolic matrices and matrix spaces. Then we describe the associated computational problems and formalize relevant concepts.

3.1 Preliminaries

The following Lemma 3.1 is a common tool which we shall need.

Lemma 3.1 (Schwartz-Zippel Lemma [Sch80; Zip79]). *Let $g \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a non-zero polynomial of degree d and $S \subseteq \mathbb{F}$. Suppose $a_1, a_2, \ldots, a_n$ are selected at random independently and uniformly from S. Then we have:*

$$\Pr[g(a_1, a_2, \ldots, a_n) = 0] \leq \frac{d}{|S|}.$$

Proof. We show that there are at most $d \cdot |S|^{n-1}$ zeroes of g in S^n. Thus it is enough to show that there are at least $(|S| - d) \cdot |S|^{n-1}$ many points in S^n on which g evaluates to non-zero. We do this by induction on n. This is obviously true for $n = 1$ because a non-zero uni-variate polynomial of degree d has at most d many zeroes in S. For the induction step, we have $g = \sum_{i=0}^{k} g_i \cdot x_n^i$ for some $k \leq d$ and $g_k \neq 0$. Here $g_i \in \mathbb{F}[x_1, x_2, \ldots, x_{n-1}]$ and $\deg(g_i) \leq d - i$. By the induction hypothesis, there are at least $(|S| - (d-k)) \cdot |S|^{n-2}$ many points $(a_1, a_2, \ldots, a_{n-1}) \in S^{n-1}$ such that $g_k(a_1, a_2, \ldots, a_{n-1}) \neq 0$. For each such $(a_1, a_2, \ldots, a_{n-1})$ we have that $g(a_1, a_2, \ldots, a_{n-1}, x_n)$ is a non-zero uni-variate polynomial of degree k in x_n. Thus for each such $(a_1, a_2, \ldots, a_{n-1})$, there are at least $(|S| - k)$ many $a_n \in S$ such that $g(a_1, a_2, \ldots, a_n) \neq 0$. Thus

$$\begin{aligned} |\{(a_1, a_2, \ldots, a_n) \mid g(a_1, a_2, \ldots, a_n) \neq 0\}| &\geq (|S| - k) \cdot (|S| - (d-k) \cdot |S|^{n-2} \\ &\geq (|S| - d) \cdot |S|^{n-1}. \end{aligned}$$

□

In the next few chapters, we shall need the following simple application of Lemma 3.1.

Lemma 3.2. *Let $g \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a non-zero polynomial of degree d such that g has a non-zero monomial of degree at most ℓ. Fix an arbitrary subset $S \subseteq \mathbb{F}$ of size $d+1$. Then there exists an assignment α to the variables x_i's with $g(\alpha) \neq 0$ such that at most ℓ variables in α are set to non-zero. Moreover this assignment α assigns these ℓ non-zero variables values from the set S.*

Proof. Let m be a non-zero monomial of g of least degree. By assumption of the lemma, we know that $\deg(m) \leq \ell$. In particular, the number of variables in m is at most ℓ. Let these variables be $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ for some $k \leq \ell$. Now we assign all the other variables to zero. This transforms g into a non-zero polynomial g' in the variables $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$. Also, $\deg(g') \leq \deg(g) = d$. Thus by using Schwartz-Zippel lemma (Lemma 3.1) we get that there exists an assignment α' to the variables $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ from set S such that $g'(\alpha') \neq 0$. By setting other variables to zero, we can extend the assignment α' to α such that $g(\alpha) \neq 0$. This assignment α obviously has all the properties claimed in the statement of the lemma. □

3.2 Matrix Spaces

We use the symbol $\mathbb{F}$ to denote the underlying field over which all the algebraic concepts are defined in this chapter. Suppose we are given m matrices $B_1, B_2, \ldots, B_m \in \mathbb{F}^{n \times n}$ and we want to determine the maximum rank of any matrix which is an $\mathbb{F}$-linear combination of $B_1, B_2, \ldots, B_m$. At this point, it is not clear why this simple looking problem is of any importance. We shall see why this problem is of fundamental importance. To this end, we formalize the following definition.

Definition 3.1 (Matrix space)**.** A vector space $\mathcal{B} \leq \mathbb{F}^{n \times n}$ is called a *matrix space*.

Thus we are given a matrix space $\mathcal{B}$ by a generating set $B_1, B_2, \ldots, B_m$ over $\mathbb{F}$. Since the dimension of $\mathbb{F}^{n \times n}$ is n^2, we can assume that $m \leq n^2$. In all the following chapters, we shall always assume that $m \leq n^2$.

3.2.1 Commutative rank

Definition 3.2. The maximum rank of any matrix in a matrix space $\mathcal{B}$ is called the *commutative rank* of $\mathcal{B}$. We write $\operatorname{crk}(\mathcal{B})$ to denote this quantity.

When the field $\mathbb{F}$ is not clear from the context, we shall use $\operatorname{crk}_{\mathbb{F}}$ to denote the rank over $\mathbb{F}$. We shall use the notation $\operatorname{rank}(A)$ for denoting the usual rank of any matrix. Note that the rank of a matrix A is same as the commutative rank of the matrix space generated by A, that is, $\operatorname{rank}(A) = \operatorname{crk}(\langle A \rangle)$. Suppose we are given a

matrix space $\mathcal{B} = \langle B_1, B_2, \ldots, B_m \rangle$. We can associate a matrix B with $\mathcal{B}$ in the following way. The matrix B has entries which are homogeneous linear forms and we define $B \stackrel{\text{def}}{=\!=} \sum_{i=1}^{m} x_i B_i$. Every matrix in $\mathcal{B}$ is the homomorphic image of B under some substitution that assigns values from $\mathbb{F}$ to the variables. This motivates the following Definition 3.3.

Definition 3.3. A matrix $B \in (\mathbb{F}[x_1, x_2, \ldots, x_m])^{n \times n}$ whose entries are homogeneous linear forms is called a *symbolic matrix.*

Now we want to compute the $\text{rank}(B)$ over the field of rational functions $\mathbb{F}(x_1, x_2, \ldots, x_m)$. This problem was introduced by Edmonds [Edm67] and is now known as Edmonds' problem. The following folklore Lemma 3.3 demonstrates a connection between the rank of symbolic matrices and matrix spaces.

Lemma 3.3. *Let* $\mathcal{B} = \langle B_1, B_2, \ldots, B_m \rangle \leq \mathbb{F}^{n \times n}$ *be a matrix space and* $B(x_1, x_2, \ldots, x_m)) = \sum_{i=1}^{m} x_i B_i$ *be the corresponding symbolic matrix. If* $|\mathbb{F}| > n$ *then* $\text{rank}(B) = \text{crk}(\mathcal{B})$.

Proof. Suppose $\lambda_1, \lambda_2, \ldots, \lambda_m \in \mathbb{F}$ are such that $\sum_{i=1}^{m} \lambda_i B_i$ is of maximum rank in $\mathcal{B}$. We have that $\text{rank}(B) \geq \text{rank}(B(\lambda_1, \lambda_2, \ldots, \lambda_m)) = \text{crk}(\mathcal{B})$. Thus $\text{rank}(B) \geq \text{crk}(\mathcal{B})$. On the other hand, suppose $r = \text{rank}(B)$. Then there exists a non-zero $r \times r$ minor M_r of $B(x_1, x_2, \ldots, x_m)$. Note that M_r is a homogeneous polynomial of degree r in $\mathbb{F}[x_1, x_2, \ldots, x_m]$. Thus by using Schwartz–Zippel Lemma (Lemma 3.1), we know that there exist $a_1, a_2, \ldots, a_m \in \mathbb{F}$ such that $M_r(a_1, a_2, \ldots, a_m) \neq 0$. Thus $\text{crk}(\mathcal{B}) \geq r = \text{rank}(\sum_{i=1}^{m} a_i B_i) = \text{rank}(B)$. Therefore $\text{rank}(B) = \text{crk}(\mathcal{B})$. □

If the field is large enough then it follows from Lemma 3.3 that computing the commutative rank of matrix spaces and computing the rank of a given symbolic matrix, are essentially the same problem. But in the cases when field is not large enough, these two notions of the rank may not be same. This is illustrated by Example 3.1.

Example 3.1. *Consider*

$$\mathcal{B} = \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\rangle$$

The corresponding symbolic matrix is:

$$B = x_1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_1 & 0 & 0 \\ 0 & x_1 + x_2 & 0 \\ 0 & 0 & x_2 \end{bmatrix}$$

Thus $\det(B) = x_1(x_1 + x_2)x_2$, *which always evaluates to zero if* $x_1, x_2 \in \mathbb{F}_2$. *Thus* $\text{crk}_{\mathbb{F}_2}(\mathcal{B}) = 2$ *whereas* $\text{rank}_{\mathbb{F}_2(x_1,x_2)}(B) = 3$ *because* $x_1(x_1 + x_2)x_2$ *is a non-zero polynomial over the rational function field* $\mathbb{F}_2(x_1, x_2)$.

To avoid this problem, we shall always assume that $|\mathbb{F}| > n$ unless stated otherwise. At first glance, the problem of commutative rank computation may appear like any other computational problem. But it is a general case of a lot of important problems in algebraic complexity theory and graph theory. The following Lemma 3.4 shows that bipartite graph matching is a special case of Edmonds' problem.

Lemma 3.4 ([Lov79]). *Let $G = (V \cup W, E)$ be a bipartite graph on $2n$ vertices. Suppose $V = \{v_1, v_2, \ldots, v_n\}, W = \{w_1, w_2, \ldots, w_n\}$ and all the edges $e \in E$ are of the form $e = (v_i, w_j)$. Let $\mathbf{X} = \{x_{11}, \ldots, x_{nn}\}$ be the set of n^2 variables. If r is the size of maximum matching in G and M_G is the following $n \times n$ symbolic matrix.*

$$(M_G)_{i,j} = \begin{cases} x_{ij} & \text{If } (v_i, w_j) \in E \\ 0 & \text{Otherwise} \end{cases}$$

Then we have $r = \text{rank}_{\mathbb{Q}(\mathbf{X})}(M_G)$.

Proof. Suppose $\{(v_{i_1}, w_{j_1}), (v_{i_2}, w_{j_2}), \ldots, (v_{i_r}, w_{j_r})\}$ is a maximum matching of G. Consider the $r \times r$ minor M_r of M_G consisting of rows $\{i_1, i_2, \ldots, i_r\}$ and columns $\{j_1, j_2, \ldots, j_r\}$. It is easy to see that the monomial $m = \prod_{k=1}^{r} x_{i_k, j_k}$ appears in M_r. Since $\{(v_{i_1}, w_{j_1}), (v_{i_2}, w_{j_2}), \ldots, (v_{i_r}, w_{j_r})\}$ are edges of G, we know that m is a non-zero monomial by the definition of M_G. Thus $r \leq \text{rank}_{\mathbb{Q}(\mathbf{X})}(M_G)$. On the other hand if $t = \text{rank}_{\mathbb{Q}(\mathbf{X})}(M_G)$ then there exists a non-zero $t \times t$ minor M_t consisting of rows rows $\{i_1, i_2, \ldots, i_t\}$ and columns $\{j_1, j_2, \ldots, j_t\}$. Since M_t is non-zero, we get that for each $a \in \{i_1, i_2, \ldots, i_t\}$ there exists a unique $b \in \{j_1, j_2, \ldots, j_t\}$ such that $(a, b) \in E$. Therefore $\{i_1, i_2, \ldots, i_t\}$ form a matching with $\{j_1, j_2, \ldots, j_t\}$. Thus $t = \text{rank}_{\mathbb{Q}(\mathbf{X})}(M_G) \leq r$. Hence $r = \text{rank}_{\mathbb{Q}(\mathbf{X})}(M_G)$. □

One can generalize Lemma 3.4 to general graphs using the so called Tutte matrix of a graph. We state this connection between the maximum matching and rank of symbolic matrices in Theorem 3.1.

Definition 3.4 (Tutte matrix [Tut47]). Tutte matrix A_G of a simple undirected graph $G = (V, E)$ with $V = [n]$ is an $n \times n$ symbolic matrix defined as below.

$$(A_G)_{i,j} = \begin{cases} x_{ij} & \text{If } (i,j) \in E \text{ and } i < j \\ -x_{ji} & \text{If } (i,j) \in E \text{ and } i > j \\ 0 & \text{Otherwise} \end{cases}$$

Theorem 3.1 ([MR95; RV89]). *If r is the size of maximum matching in G then* $\text{rank}(A_G) = 2r$.

For the proof of Theorem 3.1, we refer the reader to [RV89]. Even the so-called linear matroid parity problem is a special case of the commutative rank problem [Orl08].

Now we demonstrate some of the connections to algebraic complexity theory. For this, we first show that even if our matrix has affine linear forms as entries instead of homogeneous linear forms, it makes no difference.

Lemma 3.5. *Let $B = B_0 + x_1B_1 + \cdots + x_mB_m$ be an $n \times n$ symbolic matrix with affine linear forms in variables $x_1, x_2, \ldots, x_m$ as entries. Define $B^H \stackrel{def}{=\!=} B_0x_0 + x_1B_1 + \cdots + x_mB_m$ as the homogenized symbolic matrix corresponding to B. If $|\mathbb{F}| > n+1$ then*

$$\operatorname{rank}_{\mathbb{F}(x_1,x_2,\ldots,x_m)}(B) = \operatorname{rank}_{\mathbb{F}(x_0,x_1,x_2,\ldots,x_m)}(B^H).$$

Proof. It is clear that $\operatorname{rank}(B) \leq \operatorname{rank}(B^H)$ because setting $x_0 = 1$ can not increase the rank of $\operatorname{rank}(B^H)$. Now suppose $r = \operatorname{rank}(B^H)$. Thus there exists a non-zero $r \times r$ minor M_r of B^H. We know that M_r is a non-zero homogeneous polynomial of degree $r \leq n$ in variables $x_0, x_1, x_2, \ldots, x_m$. Since $|\mathbb{F}| > n+1$, by using the Schwartz-Zippel lemma (Lemma 3.1), we know that there exists $\lambda \neq 0$ such that $M_r(\lambda, x_1, x_2, \ldots, x_m)$ is a non-zero polynomial in $x_1, x_2, \ldots, x_m$. This also implies that $M'_r \stackrel{\text{def}}{=\!=} M_r(1, x_1, x_2, \ldots, x_m) \neq 0$, since M_r is homogeneous. Now observe that M'_r is a non-zero $r \times r$ minor of B. Thus $\operatorname{rank}(B^H) = r \leq \operatorname{rank}(B)$. Hence $\operatorname{rank}_{\mathbb{F}(x_1,x_2,\ldots,x_m)}(B) = \operatorname{rank}_{\mathbb{F}(x_0,x_1,x_2,\ldots,x_m)}(B^H)$. □

Computing the rank of a symbolic matrix has surprising connections to polynomial identity testing.

Problem 3.1 (FORMULAPIT). *Given a formula F computing $f \in \mathbb{F}[x_1, x_2, \ldots, x_m]$, is $f = 0$?*

Theorem 2.5 shows that symbolic matrix problem is useful in polynomial identity testing of polynomials computed by polynomial size formulas.

We introduce the following problem, which was called PIT in [Gar+16].

Problem 3.2 (COMMSINGULAR). *Given an $n \times n$ symbolic matrix $B \in \mathbb{F}[x_1, x_2, \ldots, x_m]$, is B of full rank, i.e., is $\operatorname{rank}(B) = n$?*

It is not hard to see that if COMMSINGULAR can be solved in deterministic polynomial time then so can be polynomial identity testing of formulas.

Lemma 3.6. *If* COMMSINGULAR $\in \mathsf{P}$ *then* FORMULAPIT $\in \mathsf{P}$.

Proof. Let $f \in \mathbb{F}[x_1, x_2, \ldots, x_m]$ be the polynomial for which we want to check if $f = 0$? We are given a formula F computing f. We compute the projection in Theorem 2.5 to construct a symbolic matrix B such that B has full rank iff $f \neq 0$. Note that the entries of B need not be homogeneous. Thus we can use Lemma 3.5 to obtain a symbolic matrix B^H with homogeneous linear forms as entries. Therefore we have the following condition.

$$f = 0 \iff \det(B) = 0 \iff \operatorname{rank}(B) < n \iff \operatorname{rank}(B^H) < n.$$

Hence the claim follows. □

In fact, one can show that the existence of a deterministic polynomial time algorithm for COMMSINGULAR implies a deterministic polynomial time algorithm for the PIT of algebraic branching programs (ABPs). ABPs are currently conjectured to be a stronger model than formulas.

Problem 3.3 (ABPPIT). *Given an ABP A computing $f \in \mathbb{F}[x_1, x_2, \ldots, x_m]$, is $f = 0$?*

In a similar vein to Lemma 3.6, the following Corollary 3.1 immediately follows by using Theorem 2.6.

Corollary 3.1. COMMSINGULAR $\in$ P *if and only if* ABPPIT $\in$ P.

Due to it being a general case of so many combinatorial and algebraic problems, it is not surprising that no efficient deterministic algorithms for COMMSINGULAR are known. Although, an easy randomized algorithm is easy to formulate which we do so below.

Problem 3.4 (COMMRANKCOMPUTE). *Given an $n \times n$ symbolic matrix $B \in \mathbb{F}[x_1, x_2, \ldots, x_m]$, compute* $\text{rank}_{\mathbb{F}(x_1, x_2, \ldots, x_m)}(B)$.

Algorithm 3.1 Randomized algorithm for COMMRANKCOMPUTE.

Input: An $n \times n$ symbolic matrix $B \in \mathbb{F}[x_1, x_2, \ldots, x_m]$ and $|\mathbb{F}| > n$.
Output: $\text{rank}_{\mathbb{F}(x_1, x_2, \ldots, x_m)}(B)$.
1: $S \leftarrow$ Any subset of size $n + 1$ of $\mathbb{F}$.
2: $\lambda_1, \lambda_2, \ldots, \lambda_m \leftarrow$ Random elements from S.
3: **return** $\text{rank}(B(\lambda_1, \lambda_2, \ldots, \lambda_m))$

Lemma 3.7. *Algorithm 3.1 computes* $\text{rank}_{\mathbb{F}(x_1, x_2, \ldots, x_m)}(B)$ *with success probability at least* $\frac{1}{n+1}$.

Proof. Suppose $r = \text{rank}_{\mathbb{F}(x_1, x_2, \ldots, x_m)}(B)$. We know that there exists a non-zero $r \times r$ minor M_r of B and M_r is a homogeneous polynomial of degree r in $x_1, x_2, \ldots, x_m$. By using the Schwartz-Zippel lemma (Lemma 3.1), we get that $M_r(\lambda_1, \lambda_2, \ldots, \lambda_m) \neq 0$ with probability at least $1 - \frac{r}{n+1} \geq \frac{1}{n+1}$. This implies that $\text{rank}(B(\lambda_1, \lambda_2, \ldots, \lambda_m)) = r$ with probability at least $\frac{1}{n+1}$. □

Note that Lemma 3.7 only succeeds with probability $\frac{1}{n+1}$. But this is not a problem because one can amplify this success probability to any desired constant by using standard probability amplification arguments. Namely, we know that Lemma 3.7 fails with probability at most $1 - \frac{1}{n+1}$. Thus if we use Lemma 3.7 independently $n + 1$ times, the failure probability is at most $\left(1 - \frac{1}{n+1}\right)^{n+1} \leq \frac{1}{e}$. Therefore by using this method, we can compute $\text{rank}_{\mathbb{F}(x_1, x_2, \ldots, x_m)}(B)$ with success probability at least $\frac{1}{e}$ and it can be further amplified to any desired constant by running Algorithm 3.1 many times independently.

Also, it is clear that Algorithm 3.1 takes $\mathsf{poly}(m,n) = \mathsf{poly}(n)$ many arithmetic operations over the field $\mathbb{F}$. It can be shown that if $\mathbb{F} = \mathbb{Q}$ then even the bit complexity of Algorithm 3.1 is $\mathsf{poly}(n)$.

We have now seen the concept of commutative rank and why computing it is of fundamental importance. There is a related notion of non-commutative rank of a matrix space or symbolic matrix. Intuitively, it means that variables $x_1, x_2, \ldots, x_m$ do not commute anymore. We now give a brief introduction to the non-commutative rank and its connection to the commutative rank.

3.2.2 Non-commutative rank

There are many equivalent ways to define the non-commutative rank of a matrix space. We start with the following definitions first and then explore some equivalent formulations. To this end, we need to define the notions of shrunk sub-spaces and discrepancy. First we define some operations between vector spaces and matrix spaces.

Definition 3.5. For any matrix A, matrix space $\mathcal{A} \leq \mathbb{F}^{n\times n}$ and vector space $U \leq \mathbb{F}^n$, we define the following linear spaces.

1. $A(U) = AU \stackrel{\text{def}}{=\!=} \{Au \mid u \in U\}$.
2. $A^{-1}(U) \stackrel{\text{def}}{=\!=} \{v \in \mathbb{F}^n \mid Av \in U\}$.
3. $\mathcal{A}(U) = \mathcal{A}U \stackrel{\text{def}}{=\!=} \langle\{Au \mid A \in \mathcal{A}, u \in U\}\rangle$.
4. $\mathcal{A}^{-1}(U) \stackrel{\text{def}}{=\!=} \cap_{A\in\mathcal{A}} A^{-1}(U) = \{v \in \mathbb{F}^n \mid \forall A \in \mathcal{A}, Av \in U\}$.

Now the properties in the following Lemma 3.8 are easy to verify.

Lemma 3.8. *For any $\mathcal{A} \leq \mathbb{F}^{n\times n}$ and vector space $U \leq \mathbb{F}^n$, the following holds.*

1. *If $U \leq W$ then $\mathcal{A}(U) \subseteq \mathcal{A}(W)$ and $\mathcal{A}^{-1}(U) \subseteq \mathcal{A}^{-1}(W)$.*
2. *$U \leq \mathcal{A}^{-1}(\mathcal{A}(U))$ and $\mathcal{A}(\mathcal{A}^{-1}(U)) \leq U$.*

Definition 3.6 (c-shrunk subspace)**.** A subspace $U \leq \mathbb{F}^n$ is called a c-shrunk subspace of a matrix space $\mathcal{B} \leq \mathbb{F}^{n\times n}$ if $\dim(U) - \dim(\mathcal{B}U) \geq c$.

Definition 3.7 (Discrepancy)**.** The discrepancy of a matrix space $\mathcal{B} \leq \mathbb{F}^{n\times n}$ is defined as the maximum c such that there exists a c-shrunk subspace of $\mathcal{B}$. Namely,

$$\mathrm{disc}(\mathcal{B}) \stackrel{\text{def}}{=\!=} \max\{c \in \mathbb{N} \mid \exists \text{ a } c\text{-shrunk subspace of } \mathcal{B}\}.$$

Definition 3.8 (Non-commutative rank)**.** The non-commutative rank $\mathrm{ncrk}(\mathcal{B})$ of a matrix space $\mathcal{B} \leq \mathbb{F}^{n\times n}$ is defined as:

$$\mathrm{ncrk}(\mathcal{B}) \stackrel{\text{def}}{=\!=} n - \mathrm{disc}(\mathcal{B}).$$

It is easy to verify the following Fact 3.1.

Fact 3.1. *For all invertible matrices $P, Q \in \mathbb{F}^{n\times n}$ and all matrix spaces $\mathcal{B} \leq \mathbb{F}^{n\times n}$, we have that* $\mathrm{crk}(\mathcal{B}) = \mathrm{crk}(P\mathcal{B}Q)$ *and* $\mathrm{ncrk}(\mathcal{B}) = \mathrm{ncrk}(P\mathcal{B}Q)$.

It is not hard to see that $\mathrm{crk}(\mathcal{B}) \leq \mathrm{ncrk}(\mathcal{B})$, as demonstrated below in Lemma 3.9.

Lemma 3.9. *For all fields $\mathbb{F}$ and for all matrix spaces $\mathcal{B} \leq \mathbb{F}^{n\times n}$,* $\mathrm{crk}(\mathcal{B}) \leq \mathrm{ncrk}(\mathcal{B})$.

Proof. Let $r = \mathrm{ncrk}(\mathcal{B})$. This means that there exists $V \leq \mathbb{F}^n$ such that $\dim(\mathcal{B}V) = \dim(V) - (n-r)$. Therefore, for all $B \in \mathcal{B}$, $\dim(BV) \leq \dim(V) - (n-r)$. Thus $\mathrm{crk}(\mathcal{B}) \leq n - (n-r) = r = \mathrm{ncrk}(\mathcal{B})$. □

It will be useful in upcoming chapters to use an alternative characterization of the non-commutative rank. For this, we need the notion of a "blow-up" of matrix spaces.

Definition 3.9 (Tensor blow-up, [IQS17b])**.** Given a matrix space $\mathcal{B} = \langle B_1, B_2, \ldots, B_m\rangle \leq \mathbb{F}^{n\times n}$, the d^{th} tensor blow-up $\mathcal{B}^{[d]} \leq \mathbb{F}^{nd\times nd}$ of $\mathcal{B}$ is defined as below.

$$\mathcal{B}^{[d]} \stackrel{\text{def}}{=\!=} \langle A_1 \otimes B_1 + \cdots + A_m \otimes B_m \mid A_i \in \mathbb{F}^{d\times d}\rangle.$$

Lemma 3.10. *For all matrix spaces $\mathcal{B} = \langle B_1, B_2, \ldots, B_m\rangle \leq \mathbb{F}^{n\times n}$ and for all $d \in \mathbb{N}^+$, we have the following inequality:*

$$\mathrm{crk}(\mathcal{B}^{[d]}) \geq d \cdot \mathrm{crk}(\mathcal{B}).$$

Proof. Suppose $\mathrm{crk}(\mathcal{B}) = r$. Thus there exist $\lambda_1, \lambda_2, \ldots, \lambda_m \in \mathbb{F}$ such that $\mathrm{rank}(A) = r$ with $A \stackrel{\text{def}}{=\!=} \lambda_1 B_1 + \lambda_2 B_2 + \cdots + \lambda_m B_m$. Now consider the matrix $A^{[d]} \in \mathcal{B}^{[d]}$ defined as below.

$$A^{[d]} \stackrel{\text{def}}{=\!=} (\lambda_1 I_d) \otimes B_1 + \cdots + (\lambda_m I_d) \otimes B_m.$$

Note that $A^{[d]}$ is an $nd \times nd$ block diagonal matrix with d blocks, each block being the $n \times n$ matrix A. Thus $\mathrm{rank}(A^{[d]}) = rd$. Hence $\mathrm{crk}(\mathcal{B}^{[d]}) \geq d \cdot \mathrm{crk}(\mathcal{B})$. □

It was proved in [IQS17b] that for large enough fields, $\mathrm{crk}(\mathcal{B}^{[d]})$ is divisible by d.

Lemma 3.11 (Lemma 5.6 in [IQS17b])**.** *Assume that $\mathbb{F}$ is an infinite field. For any matrix space $\mathcal{B} \leq \mathbb{F}^{n\times n}$, $\mathrm{crk}(\mathcal{B}^{[d]})$ is divisible by d.*

It is also known that for large enough d, $\frac{\mathrm{crk}(\mathcal{B}^{[d]})}{d}$ is equal to $\mathrm{ncrk}(\mathcal{B})$.

Theorem 3.2 ([IQS15])**.** *Assume that $\mathbb{F}$ is an infinite field. For any matrix space $\mathcal{B} \leq \mathbb{F}^{n\times n}$,*

$$\mathrm{ncrk}(\mathcal{B}) = \max\{\frac{\mathrm{crk}(\mathcal{B}^{[d]})}{d} \mid d \in \mathbb{N}^+\}.$$

Lemma 3.12. *Assume that $\mathbb{F}$ is an infinite field. For any matrix space $\mathcal{B} = \langle B_1, B_2, \ldots, B_m \rangle \leq \mathbb{F}^{n \times n}$,*

$$\text{ncrk}(\mathcal{B}) = \max\{\frac{\text{rank}_{\mathbb{F}(X_1,X_2,\ldots,X_m)}(X_1 \otimes B_1 + \cdots + X_m \otimes B_m)}{d} \mid d \in \mathbb{N}^+\}.$$

Here X_i's are $d \times d$ matrices composed of variables $x^i_{j,k}$ for $i \in [m]$ and $j, k \in [d]$.

Proof. By using Lemma 3.3, we know that $\text{crk}(\mathcal{B}^{[d]}) = \text{rank}_{\mathbb{F}(X)}(X_1 \otimes B_1 + \cdots + X_m \otimes B_m)$. Now the result follows from Theorem 3.2. □

Lemma 3.9 states that the non-commutative rank is at least as large as the commutative rank. But how large it can be compared to the commutative rank? It is known that the non-commutative rank is at most twice the commutative rank [FR04]. We shall prove this using the *r-decomposability* criterion of [FR04]. We have shown above that the the commutative rank of a matrix space is equal to the rank of the corresponding symbolic matrix over the rational function field. Can we say something similar about the non-commutative rank? It turns out that there is a similar formulation for the non-commutative rank also, but the corresponding rational function field has to be replaced by an algebraic object called the *free skew field*.

Let $\mathbb{F}\langle x_1, x_2, \ldots, x_m \rangle$ denote the the (free) algebra of non-commutative polynomials in variables $x_1, x_2, \ldots, x_m$ over $\mathbb{F}$. The (free) algebra $\mathbb{F}\langle x_1, x_2, \ldots, x_m \rangle$ is similar to the polynomial ring $\mathbb{F}[x_1, x_2, \ldots, x_m]$, the only difference is that variables $x_1, x_2, \ldots, x_m$ do not commute. To make a field out of $\mathbb{F}[x_1, x_2, \ldots, x_m]$, one just considers the field of fractions of $\mathbb{F}[x_1, x_2, \ldots, x_m]$. This is the usual field $\mathbb{F}(x_1, x_2, \ldots, x_m)$ of rational functions. We want to do the same step for $\mathbb{F}\langle x_1, x_2, \ldots, x_m \rangle$ as well.

We want to construct a "skew field of fractions", which contains $\mathbb{F}\langle x_1, x_2, \ldots, x_m \rangle$ and is a division ring, namely every non-zero element is invertible. It turns out that the construction of this "skew field of fractions" is not as simple as in the case of $\mathbb{F}[x_1, x_2, \ldots, x_m]$. There are many ways of doing this step, but only one that is universal. This was done by Amitsur in [Ami66]. We shall refer to this field by the notation $\mathbb{F}\langle\langle x_1, x_2, \ldots, x_m \rangle\rangle$, and we call it the *free skew field*. We refer the reader to [Ami66; KVV12] for a detailed and precise description of the free skew field.

There are matrix spaces where the inequality in Lemma 3.9 is actually a strict inequality. This motivates the following definition.

Definition 3.10 (Compression space)**.** A matrix space $\mathcal{B}$ is said to be a *Compression space* if $\text{ncrk}(\mathcal{B}) = \text{crk}(\mathcal{B})$.

Some special cases of matrix spaces are known to be compression spaces.

Theorem 3.3 ([AL81; EH88])**.** *If a matrix spaces space $\mathcal{C}$ is generated by two matrices, i.e., $\mathcal{C} = \langle A, B \rangle$ for some $A, B \in \mathbb{F}^{n \times n}$, then $\mathcal{C}$ is a compression space.*

Definition 3.11. Two matrix spaces $\mathcal{B}, \mathcal{C} \leq \mathbb{F}^{n\times n}$ are *said to be equivalent* if $\mathcal{C} = P\mathcal{B}Q$ for some invertible matrices $P, Q \in \mathbb{F}^{n\times n}$.

Definition 3.12 ([FR04], r-decomposable)**.** A subspace $\mathcal{B}$ of $\mathbb{F}^{n\times n}$ is said to be *r-decomposable* if it is equivalent to a subspace $\mathcal{C}$ such that all the matrices C in $\mathcal{C}$ look like below.

$$C = \begin{pmatrix} C_{11} & 0 \\ C_{21} & C_{22} \end{pmatrix}$$

Here C_{21} is of size $i \times j$ with $i + j = r$.

Theorem 3.4 (Theorem 1 in [FR04])**.** *Assume that $\mathbb{F}$ is an infinite field and let B be a symbolic $n \times n$ matrix in variables $x_1, x_2, \ldots, x_m$. Its rank in the free field $\mathbb{F}\langle\langle x_1, x_2, \ldots, x_m\rangle\rangle$ is r if and only if the corresponding matrix space $\mathcal{B}$ is r-decomposable but not $(r-1)$-decomposable*

Note that if a matrix space $\mathcal{B}$ is r-decomposable then $\mathrm{crk}(\mathcal{B}) \leq r$ because only the rows and columns of C_{21} in Definition 3.12 can contribute to the rank of C and thus also to the rank of $\mathcal{B}$. Thus Theorem 3.4 also implies Lemma 3.9.

To prove that $\mathrm{ncrk}(\mathcal{B}) \leq 2 \cdot \mathrm{crk}(\mathcal{B})$, we need Lemma 3.13 which appears in [Fla62]. First we need to state the following folklore Fact 3.2 from linear algebra.

Fact 3.2 (Folklore)**.** *Let A be an $n \times n$ with entries from the field $\mathbb{F}$. If $r = \mathrm{rank}(A)$ then there exist invertible matrices $P, Q \in \mathbb{F}^{n\times n}$ such that*

$$A = \begin{array}{c} \overbrace{}^{r \text{ columns}} \\ \left.\begin{bmatrix} 0 & 0 \\ I_r & 0 \end{bmatrix}\right\} r \text{ rows} \\ \underbrace{}_{n-r \text{ columns}} \end{array}$$

where I_r is the $r \times r$ identity matrix.

One can prove Fact 3.2 by using elementary row and column operations.

Lemma 3.13 ([Fla62])**.** *Assume that $\mathbb{F}$ is an infinite field and let $\mathcal{B} \leq \mathbb{F}^{n\times n}$ be a matrix space with $\mathrm{crk}(\mathcal{B}) = r$. Then there exist invertible matrices $P, Q \in \mathbb{F}^{n\times n}$ such that all the matrices B in $P\mathcal{B}Q$ look like below:*

$$B = \begin{array}{c} \overbrace{}^{r \text{ columns}} \\ \left.\begin{bmatrix} B_{11} & 0 \\ B_{21} & B_{22} \end{bmatrix}\right\} r \text{ rows} \\ \underbrace{}_{n-r \text{ columns}} \end{array}$$

Proof. Since $\mathrm{crk}(\mathcal{B}) = r$, by using Fact 3.2 we know that there exist invertible matrices $P, Q \in \mathbb{F}^{n\times n}$ such that $A = \begin{bmatrix} 0 & 0 \\ I_r & 0 \end{bmatrix}$ lies in $\mathcal{B}' \stackrel{\text{def}}{=\!=} P\mathcal{B}Q$. By using Fact 3.1, we know that $\mathrm{crk}(\mathcal{B}) = \mathrm{crk}(\mathcal{B}')$. We know that all the matrices $B \in \mathcal{B}'$ look like below.

$$B = \begin{array}{c} \overbrace{}^{r \text{ columns}} \\ \left.\begin{array}{c} \\ n-r \text{ rows} \end{array}\right\{ \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & \underbrace{B_{22}}_{n-r \text{ columns}} \end{bmatrix} \left.\begin{array}{c} \\ \\ \end{array}\right\} r \text{ rows} \end{array}$$

We just need to prove that B_{12} is always zero. Suppose there exists a matrix $B \in \mathcal{B}'$ where the corresponding B_{12} is not zero. Consider the set S of matrices defined as:

$$S \stackrel{\text{def}}{=\!=} \{A + tB \mid A \in \mathcal{B}', t \in \mathbb{F}\}.$$

Since $A, B \in \mathcal{B}'$, we get that $S \subseteq \mathcal{B}'$. We shall show that the assumption $B_{12} \neq 0$ implies the existence of a rank $r+1$ matrix in S. This would contradict the fact that $\text{crk}(\mathcal{B}) = \text{crk}(\mathcal{B}') = r$. Let b be a non-zero entry of B_{12}. Consider the following $(r+1) \times (r+1)$ minor M_{r+1} of a general matrix in S:

$$M_{r+1} \stackrel{\text{def}}{=\!=} \begin{bmatrix} \mathbf{u}t & bt \\ I_r + tB_{21} & \mathbf{v}t \end{bmatrix}$$

where $\mathbf{u}$ and $\mathbf{v}$ are the row (of B_{11}) and column (of B_{22}) corresponding to b. Now we observe that $\det(M_{r+1})$ is a non-zero polynomial in t of degree $r+1$. This is because bt is the only degree 1 term in $\det(M_{r+1})$ and $b \neq 0$. Thus by using the Schwartz-Zippel lemma (Lemma 3.1), we know that there exists a $\lambda \in F$ such that $\text{rank}(A + \lambda B) \geq r+1$. Therefore $B_{12} = 0$. □

Theorem 3.5 ([FR04]). *If $\mathbb{F}$ is an infinite field then for all matrix spaces $\mathcal{B} \leq \mathbb{F}^{n \times n}$, we have the following inequality:*

$$\text{crk}(\mathcal{B}) \leq \text{ncrk}(\mathcal{B}) \leq 2 \cdot \text{crk}(\mathcal{B}).$$

Proof. The inequality $\text{crk}(\mathcal{B}) \leq \text{ncrk}(\mathcal{B})$ follows from Lemma 3.9 and also from the discussion after Theorem 3.4. Suppose $\text{crk}(\mathcal{B}) = r$. We use Lemma 3.13 to find $P, Q \in \mathbb{F}^{n \times n}$ such that all the matrices B in $\mathcal{B}' \stackrel{\text{def}}{=\!=} P\mathcal{B}Q$ look like below.

$$B = \begin{array}{c} \overbrace{}^{r \text{ columns}} \\ \left.\begin{array}{c} \\ n-r \text{ rows} \end{array}\right\{ \begin{bmatrix} B_{11} & 0 \\ B_{21} & \underbrace{B_{22}}_{n-r \text{ columns}} \end{bmatrix} \left.\begin{array}{c} \\ \\ \end{array}\right\} r \text{ rows} \end{array}$$

This implies that $\mathcal{B}$ is $2r$-decomposable. Thus by using Theorem 3.4, we get that $\text{ncrk}(\mathcal{B}) \leq 2r = 2 \cdot \text{crk}(\mathcal{B})$. □

One can also show explicit examples where $\mathrm{crk}(\mathcal{B}) < \mathrm{ncrk}(\mathcal{B})$ is achieved. For instance, see Example 3.2 below.

Example 3.2. *We consider*

$$B = \begin{bmatrix} 0 & x & y \\ -x & 0 & z \\ -y & -z & 0 \end{bmatrix}$$

over $\mathbb{Q}$. *Let* $\mathcal{B}$ *be the matrix space corresponding to* B. *We have that* $\det(B) = -xyz + xyz = 0$. *Thus* $\mathrm{crk}(\mathcal{B}) < 3$ *and it is easy to see that* $\mathrm{crk}(\mathcal{B}) = 2$ *because* $\begin{bmatrix} x & y \\ 0 & z \end{bmatrix}$ *is a non-zero* 2×2 *minor. It can also be proved that* $\mathrm{ncrk}(\mathcal{B}) = 3$.

3.3 A MAX-MIN CHARACTERIZATION OF RANKS

We now propose a new characterization of both the notions of rank defined above. To this end, we first give a brief introduction to the theory of matroids. All the matroids we consider here will be assumed to be finite unless stated otherwise.

Definition 3.13 (Matroid)**.** A finite matroid M is a pair $(E, \mathcal{I})$, where E is a finite set (called the ground set) and $\mathcal{I}$ is a family of subsets of E (called the independent sets) with the following properties:

- **Non-emptiness:** The empty set is independent, *i.e.*, $\emptyset \in \mathcal{I}$.
- **Heredity:** Every subset of an independent set is independent.
- **Exchange:** If $X \in \mathcal{I}$ and $Y \in \mathcal{I}$ are two independent sets in M where $|X| > |Y|$, then there is an element $x \in X \setminus Y$ such that $Y \cup \{x\} \in \mathcal{I}$.

The **rank** of a subset X of the ground set E is the size of the largest independent subset of X. We use the notation $r(X)$ to denote the **rank** of X. We call r to be the rank function of matroid M.

Problem 3.5 (Matroid Intersection)**.** *Given two matroids* $M_1 = (E, \mathcal{I}_1)$ *and* $M_2 = (E, \mathcal{I}_2)$, *on the same ground set* E, *find the maximum cardinality common independent set* $J \in \mathcal{I}_1 \cap \mathcal{I}_2$.

The following Theorem 3.6 was proved by Edmonds in [Edm03] and it describes a min-max formulation of the maximum cardinality of the common independent set of two finite matroids.

Theorem 3.6 (Matroid Intersection theorem [Edm03])**.** *For any matroids* M_1, M_2 *on the same ground set* E *with corresponding rank functions* r_1 *and* r_2, *we have the following equality.*

$$\max_{J \in \mathcal{I}_1 \cap \mathcal{I}_2} |J| = \min_{A \subseteq E} \{r_1(A) + r_2(E \setminus A)\}.$$

Theorem 3.6 implies the following Corollary 3.2 for linear matroids.

Corollary 3.2. *Let $a_1, a_2, \ldots, a_p \in \mathbb{F}^k$ and $b_1, b_2, \ldots, b_p \in \mathbb{F}^\ell$. The maximum number s of indices $1 \leq i_1 \leq i_2 \leq \ldots \leq i_s \leq p$ such that both sets of vectors $\{a_{i_1}, a_{i_2}, \ldots, a_{i_s}\}$ and $\{b_{i_1}, b_{i_2}, \ldots, b_{i_s}\}$ are linearly independent is given by*

$$s = \min_{J \subseteq [p]} \{\dim(\langle a_j \mid j \in J\rangle) + \dim(\langle b_j \mid j \in [p] \setminus J\rangle)\}.$$

Proof. Apply Theorem 3.6 for $E = [p]$ and a set $J \subseteq [p]$ being independent in matroid M_1 if vectors in $a_1, a_2, \ldots, a_p \in \mathbb{F}^k$ corresponding to indices in J are independent. Similarly M_2 is defined by vectors $b_1, b_2, \ldots, b_p \in \mathbb{F}^\ell$. □

As an application of Corollary 3.2, we prove Lemma 3.14 which states if a matrix space is generated by rank 1 matrices then the non-commutative rank and the commutative rank are equal, that is, such matrix spaces are compression spaces.

Lemma 3.14. *Let $\mathcal{A} \leq \mathbb{F}^{n \times n}$ be a matrix space generated by rank 1 matrices. Then*

$$\operatorname{ncrk}(\mathcal{A}) = \operatorname{crk}(\mathcal{A}) = \min\{n - \dim U + \dim(\mathcal{A}U) \mid U \leq \mathbb{F}^n\}.$$

Proof. Consider a set of rank 1 matrices generating the matrix space $\mathcal{A}$: these can be written as $a_1 b_1^T, a_2 b_2^T, \ldots, a_p b_p^T$ for some $a_i, b_i \in \mathbb{F}^n$. Let s be the largest integer such that there are s linearly independent vectors among the a_i's such that the corresponding b_i's are also linearly independent. Without the loss of generality, we may assume that these are $a_1, a_2, \ldots, a_s$ and $b_1, b_2, \ldots, b_s$ respectively. Now the matrix $a_1 b_1^T + a_2 b_2^T + \ldots + a_s b_s^T$ has rank s by elementary linear algebra, and so $\operatorname{crk}(\mathcal{A}) \geq s$. On the other hand, Corollary 3.2 implies that there exists a subset $J \subseteq [p]$ such that

$$s = \dim(\langle a_j \mid j \in J\rangle) + \dim(\langle b_j \mid j \in [p] \setminus J\rangle).$$

Now define $U \stackrel{\text{def}}{=\!=} \{x \in \mathbb{F}^n \mid b_j^T x = 0 \ \forall j \in [p] \setminus J\}$. Then $\mathcal{A}U \leq \langle a_j \mid j \in J\rangle$. Therefore we have:

$$n - \dim U + \dim \mathcal{A}U \leq n - (n - \dim(\langle b_j \mid j \in [p] \setminus J\rangle)) + \dim(\langle a_j \mid j \in J\rangle) = s.$$

Hence U is an $(n - s)$-shrunk subspace for $\mathcal{A}$. Thus we have obtained the following inequalities:

$$\begin{aligned} \operatorname{crk}(\mathcal{A}) &\geq s \\ s &\geq n - \dim U + \dim \mathcal{A}U \geq \operatorname{ncrk}(\mathcal{A}) \geq \operatorname{crk}(\mathcal{A}) \end{aligned}$$

Thus it follows that

$$\operatorname{ncrk}(\mathcal{A}) = \operatorname{crk}(\mathcal{A}) = s = \min\{n - \dim U + \dim(\mathcal{A}U) \mid U \leq \mathbb{F}^n\}.$$

□

Suppose we are given a matrix space $\mathcal{A} = \langle A_1, A_2, \ldots, A_m \rangle \leq \mathbb{F}^{n \times n}$. And we want to study its non-commutative rank. For a basis $B = \{b_1, b_2, \ldots, b_n\}$ of $\mathbb{F}^n$, we define the following two linear matroids: $M_1^{\mathcal{A},B} \stackrel{\text{def}}{=\!=} (E, \mathcal{I}_1)$ and $M_2^{\mathcal{A},B} \stackrel{\text{def}}{=\!=} (E, \mathcal{I}_2)$, where $E = [m] \times [n]$. For a set $I \subseteq E$, I is independent in $\mathcal{I}_1$ if the (multi) set of vectors $S_1^I \stackrel{\text{def}}{=\!=} \{b_j \mid \exists i \in [m] : (i,j) \in I\}$ is linearly independent. Similarly, $I \subseteq E$ is independent in $\mathcal{I}_2$ if the (multi) set of vectors $S_2^I \stackrel{\text{def}}{=\!=} \{A_i b_j \mid (i,j) \in I\}$ are independent. The notions of rank r_1, r_2 in $M_1^{\mathcal{A},B}, M_2^{\mathcal{A},B}$ are defined analogously.

Now we consider the matroid intersection problem on these two matroids. First we prove the following Lemma 3.15. We use the notation $\text{MatInt}(M_1, M_2)$ to denote the solution size of the matroid intersection problem on two matroids M_1 and M_2.

Lemma 3.15. *The following equality holds for all matrix spaces $\mathcal{A} = \langle A_1, A_2, \ldots, A_m \rangle \leq \mathbb{F}^{n \times n}$ and all basis $B = \{b_1, b_2, \ldots, b_n\}$ of $\mathbb{F}^n$.*

$$\text{MatInt}(M_1^{\mathcal{A},B}, M_2^{\mathcal{A},B}) = \max_{C_1, C_2, \ldots, C_n \in \mathcal{A}} \text{rank}([C_1 b_1; C_2 b_2; \ldots; C_n b_n]).$$

Proof. Let us use r and s to denote the two quantities in the statement, that is:

$$r \stackrel{\text{def}}{=\!=} \max_{C_1, C_2, \ldots, C_n \in \mathcal{A}} \text{rank}([C_1 b_1; C_2 b_2; \ldots; C_n b_n])$$
$$s \stackrel{\text{def}}{=\!=} \text{MatInt}(M_1^{\mathcal{A},B}, M_2^{\mathcal{A},B})$$

We have $2s$ indices $i_1, i_2, \ldots, i_s, j_1, j_2, \ldots, j_s$ such that both the sets $\{b_{i_1}, b_{i_2}, \ldots, b_{i_s}\}$ and $\{A_{j_1} b_{i_1}, A_{j_2} b_{i_2}, \ldots, A_{j_s} b_{i_s}\}$ are linearly independent. Thus if we assign $C_{i_k} = A_{j_k}$ for $k \in [s]$ and assign other C_i's to be zero then we have that $\text{rank}([C_1 b_1; C_2 b_2; \ldots; C_n b_n]) = s$. Thus $r \geq s$. Now we prove the other direction. By Theorem 3.6, we know that:

$$\text{MatInt}(M_1^{\mathcal{A},B}, M_2^{\mathcal{A},B}) = \min_{J \subseteq [m] \times [n]} \{\dim(S_1^J) + \dim(S_2^{[m] \times [n] \setminus J})\}. \tag{3.1}$$

Let $J \subseteq [m] \times [n]$ be any set which achieves the minimum in Equation (3.1). Let $T \stackrel{\text{def}}{=\!=} \{j \mid \exists i \in [m] : (i,j) \in J\}$. Note that $t \stackrel{\text{def}}{=\!=} \dim(S_1^J) = |T|$. Now we define $U \stackrel{\text{def}}{=\!=} \{b_j \mid j \in [n] \setminus T\}$. Now note that $\dim(\mathcal{A}U) \leq \dim(S_2^{[m] \times [n] \setminus J}) = s - t$. Thus for any choice of $C_1, C_2, \ldots, C_n \in \mathcal{A}$, $\dim(\langle \{C_i U \mid i \in [n]\} \rangle)$ is at most $s - t$. Thus we have the following inequality:

$$r = \max_{C_1, C_2, \ldots, C_n \in \mathcal{A}} \text{rank}([C_1 b_1; C_2 b_2; \ldots; C_n b_n]) \leq s - t + t = s.$$

□

Lemma 3.16. *For all matrix spaces $\mathcal{A} = \langle A_1, A_2, \ldots, A_m \rangle \leq \mathbb{F}^{n \times n}$, we have*

$$\operatorname{ncrk}(\mathcal{A}) = \min_{B=\{b_1,b_2,\ldots,b_n\} \text{ basis of } \mathbb{F}^n} \operatorname{MatInt}(M_1^{\mathcal{A},B}, M_2^{\mathcal{A},B}).$$

Proof. For brevity, define the following:

$$r \stackrel{\text{def}}{=\!=} \operatorname{ncrk}(\mathcal{A}) \tag{3.2}$$

$$s \stackrel{\text{def}}{=\!=} \min_{B=\{b_1,b_2,\ldots,b_n\} \text{ basis of } \mathbb{F}^n} \operatorname{MatInt}(M_1^{\mathcal{A},B}, M_2^{\mathcal{A},B}) \tag{3.3}$$

We know that there exists an $(n-r)$-shrunk subspace $U \leq \mathbb{F}^n$ for $\mathcal{A}$. Let $t \stackrel{\text{def}}{=\!=} \dim(U)$. Then we have that $\dim(\mathcal{A}U) = t - (n-r)$. Let $\{u_1, u_2, \ldots, u_t\}$ be a basis of U. We extend this to a basis $B \stackrel{\text{def}}{=\!=} \{u_1, u_2, \ldots, u_t, u_{t+1}, \ldots, u_n\}$ of $\mathbb{F}^n$. For this basis B, we have $\operatorname{MatInt}(M_1^{\mathcal{A},B}, M_2^{\mathcal{A},B}) \leq n - t + t - (n-r) = r$. Thus $s \leq r$.

For the other direction, let $B = \{b_1, b_2, \ldots, b_n\}$ be any basis of $\mathbb{F}^n$ which achieves the minimum in Equation (3.3). By Theorem 3.6, we know that:

$$\operatorname{MatInt}(M_1^{\mathcal{A},B}, M_2^{\mathcal{A},B}) = \min_{J \subseteq [m] \times [n]} \{\dim(S_1^J) + \dim(S_2^{[m]\times[n]\setminus J})\}. \tag{3.4}$$

As in the proof of Lemma 3.15, let $J \subseteq [m] \times [n]$ be any set which achieves the minimum in Equation (3.4). Let $T \stackrel{\text{def}}{=\!=} \{j \mid \exists i \in [m] : (i,j) \in J\}$. Note that $t \stackrel{\text{def}}{=\!=} \dim(S_1^J) = |T|$. Now we define $U \stackrel{\text{def}}{=\!=} \{b_j \mid j \in [n] \setminus T\}$. Now note that $\dim(\mathcal{A}U) \leq \dim(S_2^{[m]\times[n]\setminus J}) = s - t$. Since $\dim(U) = n - t$, we get that U is a $n - s$ shrunk subspace of $\mathcal{A}$. Hence $r \leq s$. Therefore $r = s$. □

In light of above results, the following min-max characterization of the non-commutative rank of matrix spaces can be easily proved.

Theorem 3.7. *For all matrix spaces $\mathcal{A} = \langle A_1, A_2, \ldots, A_m \rangle \leq \mathbb{F}^{n \times n}$, we have*

$$\operatorname{ncrk}(\mathcal{A}) = \min_{B=\{b_1,b_2,\ldots,b_n\} \text{ basis of } \mathbb{F}^n} \max_{C_1,C_2,\ldots,C_n \in \mathcal{A}} \operatorname{rank}([C_1 b_1; C_2 b_2; \ldots; C_n b_n]).$$

Proof. The claimed equality follows immediately by applying Lemma 3.15 and Lemma 3.16. □

Let us now consider the following well known inequality which was first considered by John von Neumann.

Theorem 3.8 (Max-min inequality [BV04])**.** *For any function $f : X \times Y \to \mathbb{R}$, the following inequality holds.*

$$\sup_{x \in X} \inf_{y \in Y} f(x,y) \leq \inf_{y \in Y} \sup_{x \in X} f(x,y).$$

Theorem 3.8 essentially states that max-min of a function is bounded by min-max of it. Theorem 3.7 states that min-max of a rank of a matrix is equal to the non-commutative rank. So we can also ask what algebraic quantity is described by max-min of the rank of the same matrix? Theorem 3.9 demonstrates that it is equal to the commutative rank.

Theorem 3.9. For all matrix spaces $\mathcal{A} = \langle A_1, A_2, \ldots, A_m \rangle \leq \mathbb{F}^{n \times n}$, *we have*

$$\operatorname{crk}(\mathcal{A}) = \max_{C_1, C_2, \ldots, C_n \in \mathcal{A}} \min_{B = \{b_1, b_2, \ldots, b_n\} \text{ basis of } \mathbb{F}^n} \operatorname{rank}([C_1 b_1; C_2 b_2; \ldots; C_n b_n]).$$

Proof. For brevity, define the following:

$$r \stackrel{\text{def}}{=\!=} \operatorname{crk}(\mathcal{A}) \tag{3.5}$$

$$s \stackrel{\text{def}}{=\!=} \max_{C_1, C_2, \ldots, C_n \in \mathcal{A}} \min_{B = \{b_1, b_2, \ldots, b_n\} \text{ basis of } \mathbb{F}^n} \operatorname{rank}([C_1 b_1; C_2 b_2; \ldots; C_n b_n]). \tag{3.6}$$

Let $A \in \mathcal{A}$ be such that $\operatorname{rank}(A) = r$, we assign $C_1 = C_2 = \cdots = C_n = A$. Then for any basis $B = \{b_1, b_2, \ldots, b_n\}$ of $\mathbb{F}^n$, we have

$$\begin{aligned} \operatorname{rank}([C_1 b_1; C_2 b_2; \ldots; C_n b_n]) &= \operatorname{rank}([A b_1; A b_2; \ldots; A b_n]) \\ &= \operatorname{rank}(A[b_1; b_2; \ldots; b_n]) \\ &= \operatorname{rank}(A) = r \end{aligned}$$

Thus $s \geq r$.

For the other direction, let $\{C_1, C_2, \ldots, C_n\} \subseteq \mathcal{A}$ be a set of matrices which achieve the maximum in Equation (3.6). We know that for all $i \in [n]$: $\operatorname{rank}(C_i) \leq r$. Choose a vector $b_1 \in \operatorname{Ker}(C_1)$. We repeat this process for $C_2, C_3, \ldots, C_n$ as well. Namely, we choose a vector $b_2 \in \operatorname{Ker}(C_2)$ such that b_1 and b_2 are independent. More generally, we choose vectors $b_1, b_2, \ldots, b_{n-r}$ such that $b_i \in \operatorname{Ker}(C_i)$ and $\dim(b_1, b_2, \ldots, b_{n-r}) = n - r$. This can be done because the ranks of all the C_i's are at most r. And now we extend b_1, $b_2, \ldots, b_{n-r}$ to a basis $B = \{b_1, b_2, \ldots, b_n\}$ of $\mathbb{F}^n$. In this basis, we know that $C_i b_i = \mathbf{0}$ for all $i \in [n - r]$. Thus $s \leq \operatorname{rank}([C_1 b_1; C_2 b_2; \ldots; C_n b_n]) \leq r$.

Hence $s = r$. □

4

PTAS FOR COMMUTATIVE RANK

In Chapter 3, we motivated why the problem of computation of the commutative rank is an essential problem. This chapter deals with algorithms to compute the commutative rank. We also saw that the non-commutative rank is sandwiched between the commutative rank and twice that of the commutative rank. This suggests that the algorithms to compute the non-commutative rank can be used to find a $\frac{1}{2}$-approximation of the commutative rank. But the problem of computing the non-commutative rank appears even harder. Naively, even a randomized algorithm to compute the non-commutative rank seems non-trivial. Thus, we first survey the known algorithms to compute the non-commutative rank. To this end, we formalize the following problem.

Problem 4.1 (NONCOMMSINGULAR). *Given an $n \times n$ symbolic matrix $B \in \mathbb{F}[x_1, x_2, \ldots, x_m]$, is $\mathrm{ncrk}(B) = n$? This problem was called* SINGULAR *in [Gar+16].*

At first glance, it is not even clear whether NONCOMMSINGULAR is decidable. Cohn [Coh75] proved that it is decidable.

Theorem 4.1 ([Coh75]). NONCOMMSINGULAR *is decidable.*

The next step was to show a definitive time bound on the complexity of NONCOMMSINGULAR. It was shown in [CR99; Iva+15] that NONCOMMSINGULAR can be solved in deterministic exponential time.

Theorem 4.2 ([CR99; Iva+15]). NONCOMMSINGULAR *can be solved in deterministic exponential time.*

Even after [CR99; Iva+15], a polynomial time (even a randomized algorithm) algorithm remained elusive for NONCOMMSINGULAR. Finally, it was shown in [Gar+16; IQS17a] that NONCOMMSINGULAR can be solved in deterministic polynomial time.

Theorem 4.3 ([Gar+16; IQS17a]). NONCOMMSINGULAR $\in$ P.

For an excellent exposition to the problem NONCOMMSINGULAR, we refer the reader to [Gar+16; Gar+15]. Also, the tutorial [Wig17] given by Avi Wigderson at CCC17 is an excellent introduction to NONCOMMSINGULAR and much more. It was also

shown in [Gar+16; IQS17a] that we can even compute the non-commutative rank in deterministic polynomial time. Therefore it follows from Theorem 3.5 that one can compute a $\frac{1}{2}$-approximation of the commutative rank in deterministic polynomial time. It was left as an open problem in [Gar+16] whether this approximation ratio can be improved. This chapter answers this question affirmatively.

More specifically, this chapter develops an algorithm which can compute a $(1-\epsilon)$-approximation of the commutative rank in deterministic polynomial time, for any arbitrary constant $0 < \epsilon < 1$.

It is useful first to see a simple algorithm which computes a $\frac{1}{2}$-approximation of the commutative rank in deterministic polynomial time. In contrast to the algorithms described in [Gar+16; IQS17a], this algorithm is much simpler.

To bound the commutative rank of a matrix space, we also need the following easy fact from linear algebra.

Fact 4.1. *Let M be a matrix of the following form:*

$$M = \begin{array}{c} \overset{r \text{ columns}}{\overbrace{\phantom{\begin{bmatrix} L & B \\ A & \mathbf{0} \end{bmatrix}}}} \\ r \text{ rows} \left\{ \begin{bmatrix} L & B \\ A & \underbrace{\mathbf{0}} \end{bmatrix} \right. \\ \underset{n-r \text{ columns}}{\phantom{\begin{bmatrix} L & B \\ A & \mathbf{0} \end{bmatrix}}} \end{array} \left. \begin{array}{l} \\ \\ \} \, n-r \text{ rows} \end{array} \right.$$

Also, let $\operatorname{rank}(A) = a$ *and* $\operatorname{rank}(B) = b$. *Then* $\operatorname{rank}(M) \leq r + \min\{a, b\}$.

4.1 $\frac{1}{2}$-APPROXIMATION ALGORITHM FOR THE COMMUTATIVE RANK

Algorithm 4.2 computes a $\frac{1}{2}$-approximation for the commutative rank. This algorithm looks for the first matrix that increases the rank of the current matrix and stops if it does not find such a matrix. Its analysis is much easier than the general case.

Lemma 4.1. *If $|\mathbb{F}| > n$, then Algorithm 4.2 runs in polynomial time and returns a matrix $A \in \mathcal{B}$ such that* $\operatorname{rank}(A) \geq \frac{1}{2} \cdot \operatorname{crk}(\mathcal{B})$.

Proof. Let A be the matrix returned by Algorithm 4.2. Assume that A has rank r. We know that there exist non-singular matrices $P, Q \in \mathbb{F}^{n \times n}$ such that

$$PAQ = \begin{array}{c} \overset{r \text{ columns}}{\overbrace{\phantom{\begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix}}}} \\ r \text{ rows} \left\{ \begin{bmatrix} I_r & 0 \\ 0 & \underbrace{0} \end{bmatrix} \right. \\ \underset{n-r \text{ columns}}{\phantom{\begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix}}} \end{array} \left. \begin{array}{l} \\ \\ \} \, n-r \text{ rows} \end{array} \right. \tag{4.1}$$

Algorithm 4.2 Greedy algorithm for $\frac{1}{2}$-approximating the commutative rank.

Input: A matrix space $\mathcal{B} = \langle B_1, B_2, \ldots, B_m \rangle \leq \mathbb{F}^{n \times n}$, input is a list of matrices $B_1, B_2, \ldots, B_m$.
Output: A matrix $A \in \mathcal{B}$ such that $\text{rank}(A) \geq \frac{1}{2} \cdot \text{crk}(\mathcal{B})$.

```
S ← An arbitrary subset of 𝔽 of size n + 1 ▷ Rank increase is checked from this set.
A ← 0                                      ▷ A is initialized to the zero matrix.
while Rank is increasing do
    for each 1 ≤ i ≤ m do
        Check if there exists a λ ∈ S such that rank(A + λB_i) > rank(A).
        if rank(A + λB_i) > rank(A) then
            A ← A + λB_i
        end if
    end for
end while
return A
```

where I_r is the $r \times r$ identity matrix. Now consider the matrix space:

$$P\mathcal{B}Q \stackrel{\text{def}}{=\!=} \langle PB_1Q, PB_2Q, \ldots, PB_mQ \rangle.$$

This does not change anything with respect to the commutative rank. Also the way the algorithm works is not changed by such a transformation. So for the analysis, we can replace $\mathcal{B}$ by $P\mathcal{B}Q$. Consider any general matrix $A + x_1B_1 + x_2B_2 + \ldots + x_mB_m$ in $\mathcal{B}$. We decompose it as below:

$$A + x_1B_1 + x_2B_2 + \ldots + x_mB_m = \begin{array}{c} \overbrace{\quad r \text{ columns} \quad}^{} \\ r \text{ rows } \{ \begin{bmatrix} I_r + B_{11} & B_{12} \\ B_{21} & \underbrace{B_{22}}_{n-r \text{ columns}} \end{bmatrix} \} n - r \text{ rows} \end{array} \tag{4.2}$$

Here the matrices B_{11}, B_{12}, B_{21} and B_{22} have linear forms in variables $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ as their entries.

Now we claim that the bottom right matrix B_{22} is the zero matrix. Assume otherwise. Thus there exists a non-zero (s,t)-entry of the above matrix with $s, t > r$. Consider the $(r+1) \times (r+1)$ sub-matrix of $A + x_1B_1 + x_2B_2 + \ldots + x_mB_m$, obtained by adding the s^{th} row (this row comes from B_{21}) and the t^{th} column (this column comes from B_{12}) to $I_r + B_{11}$. We shall denote this sub-matrix by C. C looks like below:

$$C = \begin{pmatrix} 1+\ell_{11}(\mathbf{x}) & \ell_{12}(\mathbf{x}) & \dots & \ell_{1r}(\mathbf{x}) & b_1(\mathbf{x}) \\ \ell_{21}(\mathbf{x}) & 1+\ell_{22}(\mathbf{x}) & \dots & \ell_{2r}(\mathbf{x}) & b_2(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \ell_{r1}(\mathbf{x}) & \ell_{r2}(\mathbf{x}) & \dots & 1+\ell_{rr}(\mathbf{x}) & b_r(\mathbf{x}) \\ a_1(\mathbf{x}) & a_2(\mathbf{x}) & \dots & a_r(\mathbf{x}) & c(\mathbf{x}) \end{pmatrix}. \tag{4.3}$$

Here $\ell_{i,j}$, a_i, b_j and c are homogeneous linear forms in $\mathbf{x}$. By our choice, $c(\mathbf{x}) \neq 0$. It is not hard to see that

$$\det(C) = c(\mathbf{x}) + \text{monomials of degree at least 2.} \tag{4.4}$$

Thus there exist $\lambda \in \mathbb{F}$ and $i \in [m]$ such that $\det(C(\alpha)) \neq 0$, where α is the assignment to the variables $\mathbf{x} = (x_1, x_2, \dots, x_m)$ obtained by setting $x_k = 0$ when $k \neq i$ and $x_i = \lambda$. This follows from Lemma 3.2. Since the degree of $\det(C)$ is at most n, it also follows that we have to check at most $n+1$ values for λ to find an assignment of the type $\alpha = (0, \dots, \lambda, \dots, 0)$ such that $\det(C(\alpha)) \neq 0$. In particular, the set S contains such λ.

These choices of $i \in [m]$ and $\lambda \in \mathbb{F}$ would allow Algorithm 4.2 to find a matrix A of larger rank in line 5. Thus Algorithm 4.2 would keep finding a matrix A of larger rank as long as the matrix B_{22} is non-zero. Hence it can only stop when B_{22} is the zero matrix. Notice that this is similar to as in Lemma 3.13. If B_{22} is the zero matrix, then by using Fact 4.1 we know that $\operatorname{crk}(\mathcal{B}) \leq 2r$. Thus when Algorithm 4.2 stops, it outputs a matrix A such that $\operatorname{rank}(A) \geq \frac{1}{2} \cdot \operatorname{crk}(\mathcal{B})$.

The running time is obviously polynomial since the while loop is executed at most n times and we have to check at most $n+1$ values for λ. The bit-size of the numbers that occur in the rank check step is also polynomially bounded in the size of the entries of $B_1, B_2, \dots, B_m$. □

4.2 $\frac{2}{3}$-APPROXIMATION ALGORITHM FOR THE COMMUTATIVE RANK

We saw in the proof of Lemma 4.1 that if one analyzes the degree one monomials of the special minor C(Equation (4.5)) then one of the following conditions is satisfied:

1. The minor $\det(C)$ has degree one monomials.
2. The bottom right sub-matrix B_{22} is the zero matrix. In this case, the commutative rank of the concerned matrix space $\mathcal{B}$ can be upper bounded.

In the proof of Lemma 4.1, we only analyzed the degree one monomials of $\det(C)$, giving us a $\frac{1}{2}$-approximation for the commutative rank. This strategy extends naturally by analyzing the higher degree monomials. The following Lemma 4.2 analyzes the degree two monomials. In whatever follows, we use C to denote the following $(r+1) \times (r+1)$ matrix:

$$C = \begin{pmatrix} 1+\ell_{11}(\mathbf{x}) & \ell_{12}(\mathbf{x}) & \dots & \ell_{1r}(\mathbf{x}) & b_1(\mathbf{x}) \\ \ell_{21}(\mathbf{x}) & 1+\ell_{22}(\mathbf{x}) & \dots & \ell_{2r}(\mathbf{x}) & b_2(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \ell_{r1}(\mathbf{x}) & \ell_{r2}(\mathbf{x}) & \dots & 1+\ell_{rr}(\mathbf{x}) & b_r(\mathbf{x}) \\ a_1(\mathbf{x}) & a_2(\mathbf{x}) & \dots & a_r(\mathbf{x}) & 0 \end{pmatrix}. \tag{4.5}$$

Here ℓ_{ij}, a_i, b_i are homogeneous linear forms in variables $\mathbf{x} = (x_1, x_2, \dots, x_m)$. Note that we have assumed $c(\mathbf{x})$ to be zero in Equation (4.5) because otherwise $\det(C)$ has degree one monomials, as proved in the proof of Lemma 4.1. We shall use the symbol $L \stackrel{\text{def}}{=\!=} (\ell_{ij})$ to denote the $r \times r$ matrix defined by linear forms ℓ_{ij}'s.

Lemma 4.2. *Let C be as in Equation (4.5). Then we have:*

$$\det(C) = \left(-\sum_{i=1}^{r} a_i b_i\right) + \textit{monomials of degree at least 3}.$$

Proof. By using Laplace expansion, we know that the following equality holds for $\det(C)$:

$$\det(C) = -\sum_{1 \le i,j \le r} (-1)^{i+j} a_i b_j \det(I_r + L)_{\widehat{ij}}$$

Here $\det(I_r + L)_{\widehat{ij}}$ is the determinant of the matrix obtained from $I_r + L$ by removing i^{th} column and j^{th} row. Now observe that if $i \neq j$ then $\det(I_r + L)_{\widehat{ij}}$ has a row (also a column) composed only of homogeneous linear forms ℓ_{ij}'s, that is, there are no affine entries in this row (resp. column). Thus if $i \neq j$ then $\det(I_r + L)_{\widehat{ij}}$ has no constant term. Also the constant term of $\det(I_r + L)_{\widehat{ii}}$ (the case when $i = j$) is 1. Therefore we have:

$$\det(C) = \left(-\sum_{i=1}^{r} a_i b_i\right) + \text{monomials of degree at least 3}.$$

As in the spirit of the strategy outlined above, we have analyzed the degree two monomials of $\det(C)$ in Lemma 4.2. Suppose that there exists a choice of a_i's and b_i's such that the corresponding degree two monomial contribution $-\sum_{i=1}^{r} a_i b_i$ is non-zero. Then we can efficiently find an assignment α to the variables $\mathbf{x} = (x_1, x_2, \dots, x_m)$ such that $\det(C(\alpha)) \neq 0$. This can be done similarly as was done in the proof of Lemma 4.1. In Algorithm 4.2, we always set $m-1$ variables to zero, now we need to set $m-2$ variables to zero and then try to find an assignment to the remaining two variables such that $\det(C(\alpha)) \neq 0$. This hints us to the following Algorithm 4.3. We prove in Lemma 4.3 that Algorithm 4.3 outputs a $\frac{2}{3}$-approximation of the commutative rank. □

Algorithm 4.3 Greedy algorithm for $\frac{2}{3}$-approximating the commutative rank.

Input: A matrix space $\mathcal{B} = \langle B_1, B_2, \ldots, B_m \rangle \leq \mathbb{F}^{n \times n}$, input is a list of matrices $B_1, B_2, \ldots, B_m$.
Output: A matrix $A \in \mathcal{B}$ such that $\text{rank}(A) \geq \frac{2}{3} \cdot \text{crk}(\mathcal{B})$.

1: $S \leftarrow$ An arbitrary subset of $\mathbb{F}$ of size $n+1$ ▷ Rank increase is checked from this set.
2: $A \leftarrow 0$ ▷ A is initialized to the zero matrix.
3: **while** Rank is increasing **do**
4: **for** each pair $(i, j) \in \binom{[m]}{2}$ **do**
5: Check if there exist $\lambda, \mu \in S$ such that $\text{rank}(A + \lambda B_i + \mu B_j) > \text{rank}(A)$.
6: **if** $\text{rank}(A + \lambda B_i + \mu B_j) > \text{rank}(A)$ **then**
7: $A \leftarrow A + \lambda B_i + \mu B_j$
8: **end if**
9: **end for**
10: **end while**
11: **return** A

Lemma 4.3. *If $|\mathbb{F}| > n$, then Algorithm 4.3 runs in polynomial time and returns a matrix $A \in \mathcal{B}$ such that* $\text{rank}(A) \geq \frac{2}{3} \cdot \text{crk}(\mathcal{B})$.

Proof. Let A be the matrix returned by Algorithm 4.2. Assume that A has rank r. By a similar argument as in the proof of Lemma 4.1, we can assume that any general matrix $A + x_1 B_1 + x_2 B_2 + \ldots + x_m B_m$ in $\mathcal{B}$ can be decomposed as below:

$$A + x_1 B_1 + x_2 B_2 + \ldots + x_m B_m = \begin{array}{c} \overbrace{}^{r \text{ columns}} \\ r \text{ rows } \{ \begin{bmatrix} I_r + B_{11} & B_{12} \\ B_{21} & \underbrace{B_{22}}_{n-r \text{ columns}} \end{bmatrix} \} n - r \text{ rows} \end{array} . \tag{4.6}$$

Also, we can assume that B_{22} is the zero matrix. Otherwise we can use the proof of Lemma 4.1, to claim that it was still possible to increase the rank of A in line 5 of Algorithm 4.3. We consider a $(r+1) \times (r+1)$ minor C of $A + x_1 B_1 + x_2 B_2 + \ldots + x_m B_m$ as in Equation (4.5). By using Lemma 4.2 we know that:

$$\det(C) = \left(-\sum_{i=1}^{r} a_i b_i \right) + \text{monomials of degree at least 3.}$$

Suppose there exists a choice of a_i's and b_i's such that $(-\sum_{i=1}^{r} a_i b_i) \neq 0$. Then there exist $\lambda, \mu \in \mathbb{F}$ and $i, j \in [m]$ such that $\det(C(\alpha)) \neq 0$, where α is the assignment to the variables $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ obtained by setting $x_k = 0$ when $k \neq i, j$ and

$x_i = \lambda, x_j = \mu$. This follows from Lemma 3.2. Since the degree of $\det(C)$ is at most n, it also follows that we have to check at most $n+1$ values for λ, μ to find an assignment of the type $\alpha = (0, \ldots, \lambda, \ldots \mu, 0 \ldots, 0)$ such that $\det(C(\alpha)) \neq 0$. In particular, the set S contains such λ, μ.

In this case, line 5 of Algorithm 4.3 succeeds in finding a matrix A of bigger rank. Thus when Algorithm 4.3 terminates, for all choices of a_i's and b_i's we have that $(-\sum_{i=1}^{r} a_i b_i) = 0$. Note that a_i's are chosen from the rows of B_{21} and b_i's are chosen from the columns of B_{12}. Thus we get the following equivalence :

$$\text{For all choices of } a_i\text{'s and } b_i\text{'s : } \left(-\sum_{i=1}^{r} a_i b_i\right) = 0 \iff B_{21}B_{12} = 0.$$

Thus we can assume the condition $B_{22} = B_{21}B_{12} = 0$ to be true when Algorithm 4.3 terminates. By using the Rank-nullity theorem (Theorem 2.8), we know that

$$\operatorname{rank}(B_{21}) + \dim(\operatorname{Ker}(B_{21})) = r. \tag{4.7}$$

Since $B_{21}B_{12} = 0$, we get that $\operatorname{Im}(B_{12})$ is a subspace of $\operatorname{Ker}(B_{21})$. Thus $\operatorname{rank}(B_{12}) \leq \dim(\operatorname{Ker}(B_{21}))$. Thus Equation (4.7) implies that $\operatorname{rank}(B_{21}) + \operatorname{rank}(B_{12}) \leq r$. In particular we get that $\operatorname{rank}(B_{21}) \leq \frac{r}{2}$ or $\operatorname{rank}(B_{12}) \leq \frac{r}{2}$. Thus by using Fact 4.1, we get that $\operatorname{rank}(x_1B_1 + x_2B_2 + \ldots + x_mB_m) = \operatorname{crk}(\mathcal{B}) \leq \frac{3r}{2}$. Hence it follows that $\operatorname{rank}(A) \geq \frac{2}{3}\operatorname{crk}(\mathcal{B})$. This proves that Algorithm 4.3 computes a $\frac{2}{3}$-approximation of $\operatorname{crk}(\mathcal{B})$. The polynomial time running bound follows as in the proof of Lemma 4.1. □

4.3 $(1-\epsilon)$-APPROXIMATION ALGORITHM FOR THE COMMUTATIVE RANK

Now we have seen that analyzing the higher degree monomials of $\det(C)$ gives us a method to approximate $\operatorname{crk}(\mathcal{B})$ with a better approximation ratio. Thus our strategy considers the following two scenarios.

1. $\det(C)$ has "low" degree monomials. In this case, we can "easily" find an assignment to x_i's such that $\det(C) \neq 0$. This ensures that $\operatorname{rank}(Q(A + x_1B_1 + x_2B_2 + \ldots + x_mB_m)) > r$. This is our rank increasing step.

2. $\det(C)$ has no non-zero monomials of "low" degree. In this case, we show that $r = \operatorname{rank}(A)$ is already a good approximation of $\operatorname{crk}(\mathcal{B})$.

We have shown above how to analyze degree one and two monomials of $\det(C)$. Unfortunately, manually analyzing the higher degree terms seems to be tedious. We managed to analyze the degree three monomials manually. We remark the following equality for the degree three terms of $\det(C)$, which we shall prove later. Then we shall describe a more unified way to analyze the higher degree terms of $\det(C)$. For the sake of brevity, for the rest of this section we use the symbol **a** to denote the row vector

$\begin{bmatrix} a_1 & a_2 & \dots & a_r \end{bmatrix}$ and the symbol $\mathbf{b}$ to denote the column vector $\begin{bmatrix} b_1 & b_2 & \dots & b_r \end{bmatrix}^T$.

Claim 4.1 (Follows from results below). Let C be as in Equation (4.5). For $\det(C)$, we have the following equality:

$$\det(C) = -\mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot L \cdot \mathbf{b} - \mathbf{a} \cdot \mathbf{b} \cdot \mathrm{Tr}(L) + \text{monomials of degree at least 4.}$$

Thus if for all choices of a_i's and b_i's, degree one, two and three terms are zero then by using the ideas in the proof of Lemma 4.3, we get the following condition.

$$B_{22} = B_{21}B_{12} = B_{21}B_{11}B_{12} = 0. \tag{4.8}$$

If the above Equation (4.8) is not satisfied then it is easy to increase the rank of A as in the line 5 of Algorithm 4.3, by just making coefficients of three matrices B_i, B_j, B_k non-zero instead of two. On the other hand if Equation (4.8) is satisfied then one can prove that $\mathrm{rank}(x_1B_1 + x_2B_2 + \ldots + x_mB_m) = \mathrm{crk}(\mathcal{B}) \leq \frac{4r}{3}$, giving us an algorithm which computes a $\frac{3}{4}$-approximation of $\mathrm{crk}(\mathcal{B})$.

Now we analyze the higher degree terms of $\det(C)$ by a unified approach, instead of calculating manually like we have done till now. First we need a formulation for the adjugate (also know as the adjoint) of a matrix.

Fact 4.2. *For a square $r \times r$ matrix L, let $p_L(t) \stackrel{def}{=\!=} \det(tI - L) = \sum_{i=0}^{r} p_{r-i} \cdot t^i$ be the characteristic polynomial of L with $p_0 = 1$. Define $q_L(t) \stackrel{def}{=\!=} \frac{p_L(t) - p_L(0)}{t}$. Then we have:*

$$\mathrm{adj}(L) = (-1)^{r+1} q_L(L).$$

Theorem 4.4. *For a square $r \times r$ matrix L, let $p_L(t) \stackrel{def}{=\!=} \det(tI - L) = \sum_{i=0}^{r} p_{r-i} \cdot t^i$ be the characteristic polynomial of L with $p_0 = 1$. Then we have:*

$$\mathrm{adj}(I + L) = \sum_{j=0}^{r-1} (-1)^j L^j \sum_{i=0}^{r-j-1} (-1)^i p_i.$$

Proof. First we compute the characteristic polynomial p_{I+L} of $I + L$. We have:

$$\begin{aligned} p_{I+L}(t) &= \det(tI - (I + L)) \\ &= \det((t-1)I - L)) \\ &= p_L(t-1). \end{aligned}$$

Thus we have :

$$\begin{aligned} q_{I+L}(t) &\overset{\text{def}}{=\!=} \frac{p_{I+L}(t) - p_{I+L}(0)}{t} \\ &= \frac{p_L(t-1) - p_L(-1)}{t}. \\ &= \sum_{i=0}^{r} \frac{p_{r-i} \cdot ((t-1)^i - (-1)^i)}{t} \\ &= \sum_{i=1}^{r} p_{r-i} \cdot \left(\sum_{j=0}^{i-1} (-1)^j (t-1)^{i-j-1} \right). \end{aligned}$$

Therefore

$$\begin{aligned} \text{adj}(I+L) &= (-1)^{r+1} q_{I+L}(I+L) \\ &= (-1)^{r+1} \sum_{i=1}^{r} p_{r-i} \cdot \left(\sum_{j=0}^{i-1} (-1)^j (L)^{i-j-1} \right) \\ &= \sum_{j=0}^{r-1} (-1)^j L^j \cdot \left(\sum_{i=0}^{r-j-1} (-1)^i p_i \right). \end{aligned}$$

□

Lemma 4.4. *Let* $p_L(t) \overset{def}{=\!=} \det(tI - L) = \sum_{i=0}^{r} p_{r-i} \cdot t^i$ *be the characteristic polynomial of* L *with* $p_0 = 1$. *Then we have the following equality:*

$$\det(C) = -\mathbf{a} \cdot \left(\sum_{j=0}^{r-1} (-1)^j L^j \sum_{i=0}^{r-j-1} (-1)^i p_i \right) \cdot \mathbf{b}.$$

Proof. By using Laplace expansion, we know that the following equality holds for $\det(C)$:

$$\det(C) = -\sum_{1 \le i,j \le r} (-1)^{i+j} a_i b_j \det(I_r + L)_{\widehat{ij}}$$

Here $\det(I_r + L)_{\widehat{ij}}$ is the determinant of the sub-matrix obtained from $I_r + L$ by removing the i^{th} column and the j^{th} row. We also know that $\det(I+L)_{\widehat{ij}} = (-1)^{i+j}(\text{adj}(I+L))_{ij}$. Thus

$$\begin{aligned} \det(C) &= -\sum_{i,j} (-1)^{i+j} a_i b_j (-1)^{i+j} (\text{adj}(I+L))_{ij} \\ &= -\sum_{i,j} a_i b_j (\text{adj}(I+L))_{ij} \\ &= -\mathbf{a} \cdot \text{adj}(I+L) \cdot \mathbf{b}. \end{aligned}$$

Now the result follows from Theorem 4.4. □

Lemma 4.4 allows us to easily analyze the higher degree terms of $\det(C)$.

Lemma 4.5. *Let C be as in Equation (4.5). Let $k \in \{-1, 0, \ldots, r-1\}$ be the maximum integer such that $\mathbf{a} \cdot L^i \cdot \mathbf{b}$ is zero for all $i \in \{0, 1, \ldots, k\}$ (If no such i exists then we set $k = -1$) and let $m \in \{2, 3, \ldots, r+1\}$ be the least integer such that $\det(C)$ has a non-zero monomial of degree m. Then $m = k + 3$.*

Proof. Note that all the monomials in $\det(C)$ are of degree at least two. We note that the entries of L^j are homogeneous polynomials in variables $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ of degree j.

If $p_L(t) = \det(tI - L) = p_0 t^r + p_1 t^{r-1} + \cdots + p_r$ is the characteristic polynomial of L then we know that p_j is also a homogeneous polynomial in variables $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ of degree j. Therefore it follows that the sum D_s of all degree s monomials in $\det(C)$ is:

$$\begin{aligned} D_s &\stackrel{\text{def}}{=\!=} -\mathbf{a} \cdot \sum_{j=0}^{s-2} (-1)^j L^j (-1)^{s-2-j} p_{s-2-j} \cdot \mathbf{b} \\ &= (-1)^{s-1} \sum_{j=0}^{s-2} p_{s-2-j} \cdot \mathbf{a} \cdot L^j \cdot \mathbf{b} \end{aligned}$$

We have that $\mathbf{a} \cdot L^i \cdot \mathbf{b}$ is zero for all $i \in \{0, 1, \ldots, k\}$ but $\mathbf{a} \cdot L^{k+1} \cdot \mathbf{b} \neq 0$. This implies that $D_2 = D_3 = \cdots = D_{k+2} = 0$. Also, we have:

$$D_{k+3} = (-1)^{k+2} \cdot \mathbf{a} \cdot L^{k+1} \cdot \mathbf{b} \neq 0.$$

Hence $m = k + 3$. □

Now the strategy to find an arbitrary approximation of the commutative rank is straight forward. If $\det(C)$ has "small" degree monomials then extend the Algorithm 4.3 such that now we check with a linear combination of "few" matrices, whether the rank can be increased. Otherwise we get some conditions from Lemma 4.5 which can be used to upper bound the commutative rank. To this end, we prove the following Lemma 4.6.

Lemma 4.6. *Let $\mathbb{F}$ be any field, $B \in \mathbb{F}^{n \times n}$ and*

$$B = \begin{array}{c} \overbrace{\phantom{r \text{ columns}}}^{r \text{ columns}} \\ r \text{ rows } \{ \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \} n - r \text{ rows} \\ \underbrace{\phantom{n - r \text{ columns}}}_{n - r \text{ columns}} \end{array} \tag{4.9}$$

Consider the sequence of matrices $B_{22}, B_{21}B_{12}, B_{21}B_{11}B_{12}, \ldots, B_{21}B_{11}^{j}B_{12} \ldots$. *If the first* $k \geq 1$ *matrices in this sequence are equal to the zero matrix and* B_{11} *is non-singular, then* $\operatorname{rank}(B) \leq r\left(1+\frac{1}{k}\right)$.

Proof. If $\operatorname{rank}(B_{12}) \leq \frac{r}{k}$, then we are done by using the Fact 4.1. So we can assume without loss of generality that $\operatorname{rank}(B_{12}) > \frac{r}{k}$. Now suppose that

$$\dim\langle \operatorname{Im}(B_{12}) \cup \operatorname{Im}(B_{11}B_{12}) \cup \ldots \cup \operatorname{Im}(B_{11}^{k-2}B_{12})\rangle \geq (k-1)\operatorname{rank}(B_{12}).$$

We note that $\operatorname{Im}(B_{12}), \operatorname{Im}(B_{11}B_{12}), \ldots, \operatorname{Im}(B_{11}^{k-2}B_{12})$, are sub-spaces of $\operatorname{Ker}(B_{21})$. Further using the Rank-nullity theorem (Theorem 2.8), we get $\operatorname{rank}(B_{21}) < r - \frac{r\cdot(k-1)}{k} = \frac{r}{k}$. By using Fact 4.1, we again get that $\operatorname{rank}(B) \leq r\left(1+\frac{1}{k}\right)$.

In the above discussion, we assumed that

$$\dim\langle \operatorname{Im}(B_{12}) \cup \operatorname{Im}(B_{11}B_{12}) \cup \ldots \cup \operatorname{Im}(B_{11}^{k-2}B_{12})\rangle \geq (k-1)\operatorname{rank}(B_{12}).$$

What if this is not the case? We still want to use the same idea as above but we want to ensure this assumption. For this purpose, we use a series of elementary column operations on B to transform it to a new matrix B^*, which satisfies the above assumption. Since the rank of a matrix is invariant under elementary column operations, we obtain the desired rank bound. Now we show how to obtain this matrix B^* using a series of elementary column operations on B. Whenever we apply these elementary column operations on B, we also maintain the invariant that $B_{22} = B_{21}B_{12} = B_{21}B_{11}B_{12} = \ldots = B_{21}B_{11}^{k-2}B_{12} = 0$.

Suppose

$$\dim\langle \operatorname{Im}(B_{12}) \cup \operatorname{Im}(B_{11}B_{12}) \cup \ldots \cup \operatorname{Im}(B_{11}^{k-2}B_{12})\rangle < (k-1)\operatorname{rank}(B_{12}). \tag{4.10}$$

Let $\rho := \operatorname{rank}(B_{12})$. First, we can assume that B_{12} has exactly ρ non-zero columns. This can be achieved by performing elementary column operations on the last $n-r$ columns. This does not change the matrix $B_{22} = 0$. Furthermore, these column operations correspond to replacing B_{12} by $B_{12}\cdot S$ for some invertible $(n-r)\times(n-r)$-matrix S. Since $B_{22} = B_{21}B_{12} = B_{21}B_{11}B_{12} = \ldots = B_{21}B_{11}^{k-2}B_{12} = 0$ implies $B_{21}B_{12}S = B_{21}B_{11}B_{12}S = \ldots = B_{21}B_{11}^{k-2}B_{12}S = 0$, so we maintain our invariant. We will call the new matrix again B_{12}.

Note that the image of a matrix is its column span. Since every matrix $B_{11}^{i}B_{12}$ has exactly ρ non-zero columns (since B_{12} has ρ non-zero columns and B_{11} is non-singular), assumption in Equation (4.10) means that there is a linear dependence between these columns. That means there are vectors $y_0, y_1, \ldots, y_{k-2} \in \mathbb{F}^{n-r}$, not all equal to zero, such that $\sum_{i=0}^{k-2} B_{11}^{i}B_{12}\cdot y_i = 0$. Moreover, we can assume that these vectors only have non-zero entries in the places that corresponds to non-zero columns of B_{12}. First we show that we can assume $y_0 \neq 0$. Suppose $0 \leq j \leq k-2$ is the least integer such that $y_j \neq 0$. So we left multiply the equation $\sum_{i=0}^{k-2} B_{11}^{i}B_{12}\cdot y_i = 0$ by $(B_{11}^{j})^{-1}$, giving us

$(B_{11}^j)^{-1} \sum_{i=0}^{k-2} B_{11}^i B_{12} \cdot y_i = \sum_{i=j}^{k-2} B_{11}^{i-j} B_{12} \cdot y_i = 0$. By renumbering the indices, this can be re-written as $\sum_{i=0}^{k-2-j} B_{11}^i B_{12} \cdot y_i = 0$. Thus we can assume that $y_0 \neq 0$. (The new sum runs only up to $k-2-j$, for the missing summands, we choose the corresponding y_i to be zero.)

By writing $\sum_{i=0}^{k-2} B_{11}^i B_{12} \cdot y_i = 0$ as $B_{12} \cdot y_0 + B_{11} \cdot \sum_{i=1}^{k-2} B_{11}^{i-1} B_{12} y_i = 0$, we see that there is a linear dependence between the columns of B_{12} and B_{11}. Let $\ell \in [n-r]$ be such that ℓ^{th} entry of y_0 is non-zero. Therefore, we can make the ℓ^{th} column of B_{12} zero by adding a multiple of $\sum_{i=1}^{k-2} B_{11}^i B_{12} \cdot y_i$ and maybe adding some multiple of some other columns of B_{12} to it. This will decrease the rank of B_{12} by 1.

We claim that our invariant is still fulfilled. First, we add $B_{11} \cdot \sum_{i=1}^{k-2} B_{11}^{i-1} B_{12} \cdot y_i$ to the ℓ^{th} column of B_{12} and this will also add $B_{21} \cdot \sum_{i=1}^{k-2} B_{11}^{i-1} B_{12} \cdot y_i$ to the ℓ^{th} column of B_{22}. Since the invariant was fulfilled before the operation, B_{22} will stay zero. As seen before, column operations within the last $n-r$ columns do not change B_{22}. Thus, one of the $n-r$ columns on the right-hand side (namely the side composed of B_{12} and B_{22}) of B became zero. We can remove this column from our consideration. Let B' and B'_{12} the matrices obtained from B and B_{12} by removing this zero column. Since the columns of B'_{12} are a subset of the columns of B_{12}, $B_{21}B_{12} = B_{21}B_{11}B_{12} = \ldots = B_{21}B_{11}^{k-2}B_{12} = 0$ implies that $B_{21}B'_{12} = B_{21}B_{11}B'_{12} = \cdots = B_{21}B_{11}^{k-2}B'_{12} = 0$. Therefore, our invariant is still valid.

We repeat this process until the Equation (4.10) is not true anymore. Note that this happens for sure when $\operatorname{rank}(B_{12}) = 0$. At the end of this process we get a matrix B^* such that

$$\dim\langle \operatorname{Im}(B_{12}^*) \cup \operatorname{Im}(B_{11}B_{12}^*) \cup \ldots \cup \operatorname{Im}(B_{11}^{k-2}B_{12}^*)\rangle \geq (k-1)\operatorname{rank}(B_{12}^*).$$

Now the rank bound follows from the argument given above. □

Now we are ready to give the final algorithm.

The following theorem proves the correctness of Algorithm 4.4. Let s be an upper bound on the bit size of the entries of $B_1, \ldots, B_m$.

Theorem 4.5. *Assume that $|\mathbb{F}| > n$. Algorithm 4.4 runs in time $O((mn)^{\frac{1}{\epsilon}} \cdot M(n, s+\log n) \cdot n)$ and returns a matrix $A \in \mathcal{B}$ such that $\operatorname{rank}(A) \geq (1-\epsilon) \cdot \operatorname{crk}(\mathcal{B})$, where $M(n,t)$ is the time required to compute the rank of an $n \times n$ matrix with entries of bit size at most t.*

Proof. Let A be the matrix returned by Algorithm 4.4. Assume that A has rank r. We know that there exist non-singular matrices $P, Q \in \mathbb{F}^{n\times n}$ such that

Algorithm 4.4 Greedy algorithm for $(1-\epsilon)$-approximating commutative rank

Input: A matrix space $\mathcal{B} = \langle B_1, B_2, \ldots, B_m \rangle \leq \mathbb{F}^{n\times n}$, input is a list of matrices $B_1, B_2, \ldots, B_m$. An approximation parameter $0 < \epsilon < 1$ and $|\mathbb{F}| > n$.

Output: A matrix $A \in \mathcal{B}$ such that $\text{rank}(A) \geq (1-\epsilon)\cdot \text{crk}(\mathcal{B})$.

1: $S \leftarrow$ An arbitrary subset of $\mathbb{F}$ of size $n+1$ ▷ Rank increase is checked from this set.
2: $\ell \leftarrow \lceil \frac{1}{\epsilon} - 1 \rceil$ ▷ ℓ = Number of matrices with which linear combination has to be checked.
3: $A \leftarrow 0$ ▷ A is initialized to the zero matrix.
4: **while** Rank is increasing **do**
5: **for** each $\{i_1, i_2, \ldots, i_\ell\} \in \binom{[m]}{\ell}$ **do** ▷ This means we try all combinations of matrices $B_{i_1}, B_{i_2}, \ldots, B_{i_\ell}$.
6: Check if there exist $\lambda_1, \lambda_2, \ldots, \lambda_\ell \in S$ such that $\text{rank}(A + \lambda_1 B_{i_1} + \lambda_2 B_{i_2} + \ldots + \lambda_\ell B_{i_\ell}) > \text{rank}(A)$.
7: **if** $\text{rank}(A + \lambda_1 B_{i_1} + \lambda_2 B_{i_2} + \ldots + \lambda_\ell B_{i_\ell}) > \text{rank}(A)$ **then**
8: $A \leftarrow A + \lambda_1 B_{i_1} + \lambda_2 B_{i_2} + \ldots + \lambda_\ell B_{i_\ell}$
9: **end if**
10: **end for**
11: **end while**
12: **return** A

$$PAQ = \begin{array}{c} \overset{r \text{ columns}}{\overbrace{}} \\ r \text{ rows} \left\{ \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} \right\} n-r \text{ rows} \\ \underbrace{}_{n-r \text{ columns}} \end{array} \tag{4.11}$$

where I_r is the $r \times r$ identity matrix. Now consider the matrix space

$$P\mathcal{B}Q \stackrel{\text{def}}{=\!=} \langle PB_1Q, PB_2Q, \ldots, PB_mQ \rangle.$$

This does not change anything with respect to the rank. Also the way the algorithm works is not changed by such a transformation. So for the analysis, we can replace $\mathcal{B}$ by $P\mathcal{B}Q$. Consider any general matrix $A + x_1B_1 + x_2B_2 + \ldots + x_mB_m$ in $\mathcal{B}$. As usual, we decompose it as below:

$$A + x_1B_1 + x_2B_2 + \ldots + x_mB_m \overset{r \text{ rows}}{=} \overbrace{\left\{\begin{bmatrix} I_r + B_{11} & B_{12} \\ B_{21} & \underbrace{B_{22}}_{n-r \text{ columns}} \end{bmatrix}\right\}}^{r \text{ columns}} n - r \text{ rows} \tag{4.12}$$

Now we claim that the first ℓ matrices in the sequence B_{22}, $B_{21}B_{12}$, $B_{21}B_{11}B_{12}$, ..., $B_{21}B_{11}^jB_{12}$, ... are equal to the zero matrix. Assume otherwise. If $B_{22} \neq 0$ then line 6 will obviously increase the rank of A and thus Algorithm 4.4 could not have terminated with A. This follows from exactly the same argument as in the proof of Lemma 4.1. Thus $B_{22} = 0$. Therefore our assumption implies that there exists $i \in \{0, 1, \ldots, \ell - 2\}$ such that $B_{21}B_{11}^iB_{12} \neq 0$. Thus there exists a row $\mathbf{a}$ of B_{21} and a column $\mathbf{b}$ of B_{12} such that $\mathbf{a}B_{11}^i\mathbf{b} \neq 0$. Now we use Lemma 4.5 with $B_{11} = L$. Thus $\det(C)$ in Equation (4.5) has non-zero monomials of degree at most $\ell - 3 + 3 = \ell$. In this case, by using Lemma 3.2, we know that there exists an assignment α to the x_i's such that $\det(C) \neq 0$. Moreover, Lemma 3.2 guarantees that at most ℓ of the x_i's are set to non-zero. Therefore line 6 will be able to increase the rank of A and thus Algorithm 4.4 could not have terminated with A. Therefore our assumption is false. Thus, the first ℓ matrices in the sequence $B_{22}, B_{21}B_{12}, B_{21}B_{11}B_{12}, \ldots, B_{21}B_{11}^jB_{12} \ldots$ are equal to the zero matrix. Now we apply Lemma 4.6 to get that $\operatorname{crk}(\mathcal{B}) = \operatorname{rank}(B) \leq r\left(1 + \frac{1}{\ell}\right)$. Since $\ell \geq \frac{1}{\epsilon} - 1$, we get that $1 + \frac{1}{\ell} \leq 1 + \frac{\epsilon}{1-\epsilon} = \frac{1}{1-\epsilon}$. Thus $r = \operatorname{rank}(A) \geq (1 - \epsilon)\operatorname{crk}(\mathcal{B})$.

The desired running time can be proved easily. The outer while loop runs at most n times, thus the total running time is at most n times the running time of one iteration. One iteration of the outer loop has $\binom{[m]}{\ell} = O(m^{\frac{1}{\epsilon}})$ iterations of the inner for loop. By using the Schwartz-Zippel lemma (Lemma 3.1), one iteration of inner for loop needs to try at most $(n+1)^\ell = O(n^{\frac{1}{\epsilon}})$ possible values of $\lambda_1, \lambda_2, \ldots, \lambda_\ell \in \mathbb{F}$. And then we perform two instances of rank computation. The stated running time follows. □

Remark 4.1. Algorithm 4.4 runs in time $O((mn)^{\frac{1}{\epsilon}} \cdot n \cdot M(n))$ in the algebraic RAM model. Here $M(n)$ is the time required to compute the rank of an $n \times n$ matrix in the algebraic RAM model. It is known that $M(n) = O(n^\omega)$ with ω being the exponent of matrix multiplication. Since one can assume that $m \leq n^2$, Algorithm 4.4 runs in time $O(n^{\frac{3}{\epsilon}+\omega+1})$ in algebraic RAM model.

Remark 4.2. With a more refined analysis, it can be seen that Algorithm 4.4 uses $O((mn)^{\frac{1}{\epsilon}} \cdot n \cdot M(n, s + \log n))$ bit operations if the entries of the input matrices $B_1, B_2, \ldots, B_m$ have bit size at most s. Here $M(n, t)$ is the bit complexity of computing the rank of a matrix whose entries have bit size at most t. The additional $\log n$ in the bit size comes from the fact that the entries of the final matrix A are by a polynomial factor (in n) larger than the entries of the B_i due to the update steps.

4.4 Wong sequences and Wong index

We described above a PTAS for computing the commutative rank of matrix spaces. In this section, we describe an alternative approach to prove that Algorithm 4.4 is a PTAS for computing the commutative rank. This approach was used in our paper [BJP18]. To this end, we introduce the notion of Wong sequences. For a more comprehensive exposition, we refer the reader to [Iva+15].

Definition 4.1 (Second Wong Sequence)**.** Let $\mathcal{B} \leq \mathbb{F}^{n\times n}$ be a matrix space and $A \in \mathcal{B}$. The sequence of sub-spaces $(W_i)_{i\in\mathbb{N}}$ of $\mathbb{F}^n$ is called the *second Wong sequence* of $(A, \mathcal{B})$, where $W_0 = \{0\}$, and $W_{i+1} = \mathcal{B}A^{-1}(W_i)$.

In [Iva+15], first Wong sequences are also introduced. But for our purpose, just the notion of *second Wong sequence* is enough. The following Lemma 4.7 is easy to prove.

Lemma 4.7 (Proposition 7 in [Iva+15])**.** *Let $(W_i)_{i\in\mathbb{N}}$ be the second Wong sequence of $(A, \mathcal{B})$. Then following holds.*

1. *$W_i \leq W_{i+1}$ for all $i \in \mathbb{N}$. Also, $W_i = W_{i+1}$ if and only if $A^{-1}(W_i) \leq \mathcal{B}^{-1}(W_i)$.*
2. *There exists a limit subspace W^* of $(W_i)_{i\in\mathbb{N}}$. Also, the limit subspace $W^* = W_k$ is realized by some $k \leq n$.*
3. *The limit subspace W^* is the smallest subspace T of $\mathbb{F}^n$ such that $A^{-1}(T) \leq \mathcal{B}^{-1}(T)$.*

Proof. We prove $W_i \leq W_{i+1}$ by induction on i. It trivially holds true for $i = 0$ because $W_0 = \{0\}$. By induction hypothesis we have that $W_{i-1} \leq W_i$. This implies that $W_i = \mathcal{B}A^{-1}(W_{i-1}) \leq \mathcal{B}A^{-1}(W_i) = W_{i+1}$. Suppose $A^{-1}(W_i) \leq \mathcal{B}^{-1}(W_i)$ for some W_i. Then $W_{i+1} = \mathcal{B}A^{-1}(W_i) \leq \mathcal{B}\mathcal{B}^{-1}(W_i) \leq W_i$. Thus $W_i = W_{i+1}$. For the other direction, assume that $W_i = W_{i+1} = \mathcal{B}A^{-1}(W_i)$. Now we apply the second part of Lemma 3.8 with $U = A^{-1}(W_i)$ and $\mathcal{A} = \mathcal{B}$. Thus $A^{-1}(W_i) \leq \mathcal{B}^{-1}(\mathcal{B}A^{-1}(W_i)) = \mathcal{B}^{-1}(W_{i+1}) = \mathcal{B}^{-1}(W_i)$.

Since W_i is a subspace of W_{i+1}, we get that $\dim(W_i) \leq \dim(W_{i+1})$. If $\dim(W_i) = \dim(W_{i+1})$ then obviously $W^* = W_i$. Also, the case of strict inequality $\dim(W_i) < \dim(W_{i+1})$ can only occur at most n times because $(W_i)_{i\in\mathbb{N}}$ are sub-spaces of $\mathbb{F}^n$. Thus W^* exists and is realized as $W^* = W_k$ by some $k \leq n$.

Consider an arbitrary $T \leq \mathbb{F}^n$ such that $A^{-1}(T) \leq \mathcal{B}^{-1}(T)$. By induction on i, we first show that $W_i \leq T$ for all $i \in \mathbb{N}$. It trivially holds true for $i = 0$ because $W_0 = \{0\}$. By induction hypothesis we have that $W_{i-1} \leq T$. Thus $W_i = \mathcal{B}A^{-1}(W_{i-1}) \leq \mathcal{B}A^{-1}(T)$. Since $A^{-1}(T) \leq \mathcal{B}^{-1}(T)$, we get that $\mathcal{B}A^{-1}(T) \leq \mathcal{B}\mathcal{B}^{-1}(T) \leq T$. Thus $W_i \leq T$. In particular if a subspace $T \leq \mathbb{F}^n$ satisfies $A^{-1}(T) \leq \mathcal{B}^{-1}(T)$ then $W^* \leq T$. The first item of this lemma also proves that $A^{-1}(W^*) \leq \mathcal{B}^{-1}(W^*)$. Thus W^* is the smallest subspace T of $\mathbb{F}^n$ such that $A^{-1}(T) \leq \mathcal{B}^{-1}(T)$. □

Lemma 4.7 implies that it is enough the consider the second Wong sequence $(W_i)_{i\in\mathbb{N}}$ till $i \leq n$. Thus from now on, we shall consider the second Wong sequence only for $i \leq n$.

Next, we introduce the notion of pseudo-inverses. They are helpful in computing the Wong sequences. We remark that we need the notion of Wong sequence only for the analysis, our algorithm is completely oblivious to Wong sequences.

Definition 4.2 (Pseudo-Inverse)**.** A non-singular matrix $A' \in \mathbb{F}^{n\times n}$ is called a *pseudo-inverse* of a matrix $A \in \mathbb{F}^{n\times n}$ if the restriction of A' to $\mathrm{Im}(A)$ is the inverse of the restriction of A to a direct complement of $\mathrm{Ker}(A)$.

Unlike the usual inverse of a non-singular matrix, a pseudo-inverse of a matrix is not necessarily unique. But it always exists and if A is non-singular, then it is unique and coincides with the usual inverse.

The following lemma demonstrates the role of pseudo-inverses in computing Wong sequences. This lemma and its proof are implicit in the proof of Lemma 10 in [Iva+15]. We prove it here for the sake of completeness. The lemma essentially states that we can replace the pre-image computation in the Wong sequence by multiplication with a pseudo-inverse.

Lemma 4.8. *Let $\mathbb{F}$ be any field and $\mathcal{B} \leq \mathbb{F}^{n\times n}$ be a matrix space, $A \in \mathcal{B}$, A' be a pseudo-inverse of A and $(W_i)_{i\in[[n]]}$ be the second Wong sequence of $(A, \mathcal{B})$. Then for all $1 \leq i \leq n$, we have $W_i = (\mathcal{B}A')^i(\mathrm{Ker}(AA'))$ as long as $W_{i-1} \subseteq \mathrm{Im}(A)$.*

Proof. We prove the statement by induction on i. Since $\mathrm{Ker}(AA') = (A')^{-1}(\mathrm{Ker}(A))$, we get that

$$(\mathcal{B}A')(\mathrm{Ker}(AA')) = \mathcal{B}A'(A')^{-1}(\mathrm{Ker}(A)) = \mathcal{B}\,\mathrm{Ker}(A) = W_1.$$

This proves the base case of $i = 1$. To prove that $W_i = (\mathcal{B}A')^i(\mathrm{Ker}(AA'))$, we need to prove that $(\mathcal{B}A')^i(\mathrm{Ker}(AA')) \subseteq W_i$ and $W_i \subseteq (\mathcal{B}A')^i(\mathrm{Ker}(AA'))$. By the induction hypothesis, we just need to prove that $(\mathcal{B}A')(W_{i-1}) \subseteq W_i$ and $W_i \subseteq (\mathcal{B}A')(W_{i-1})$.

First we prove the easy direction, that is $(\mathcal{B}A')(W_{i-1}) \subseteq W_i$. Since $W_{i-1} \subseteq \mathrm{Im}(A)$, we have that $A'(W_{i-1}) \subseteq A^{-1}(W_{i-1})$. Thus $(\mathcal{B}A')(W_{i-1}) \subseteq \mathcal{B}A^{-1}(W_{i-1}) = W_i$.

Now we prove that $W_i \subseteq (\mathcal{B}A')(W_{i-1})$. Since $W_{i-1} \subseteq \mathrm{Im}(A)$, we get that $A^{-1}(W_{i-1}) = A'W_{i-1} + \mathrm{Ker}(A)$. Thus $W_i = \mathcal{B}A^{-1}(W_{i-1}) \subseteq \mathcal{B}A'W_{i-1} + \mathcal{B}\,\mathrm{Ker}(A)$. We have $\mathcal{B}\,\mathrm{Ker}(A) = W_1 \subseteq W_{i-1}$, this implies that $W_i \subseteq \mathcal{B}A'W_{i-1} + W_{i-1}$. Since $A \in \mathcal{B}$ and $W_{i-1} = AA'W_{i-1}$, we get that $W_{i-1} \subseteq \mathcal{B}A'W_{i-1}$. This in turn implies that $W_i \subseteq \mathcal{B}A'W_{i-1} + \mathcal{B}A'W_{i-1} = (\mathcal{B}A')(W_{i-1})$. □

Given a matrix space $\mathcal{B}$ and a matrix $A \in \mathcal{B}$, how can one check that A is of maximum rank in $\mathcal{B}$, *i.e.*, $\mathrm{rank}(A) = \mathrm{crk}(\mathcal{B})$? The following lemma in [Iva+15] gives a sufficient condition for A to be of maximum rank in $\mathcal{B}$.

Lemma 4.9 (Lemma 10 in [Iva+15]). *Assume that $|\mathbb{F}| > n$. Let $A \in \mathcal{B} \leq \mathbb{F}^{n\times n}$, and let A' be a pseudo-inverse of A. If we have that for all $i \in [n]$,*

$$W_i = (\mathcal{B}A')^i(\mathrm{Ker}(AA')) \subseteq \mathrm{Im}(A), \tag{4.13}$$

then A is of maximum rank in $\mathcal{B}$.

Thus, Lemma 4.9 shows that if A is not of maximum rank in $\mathcal{B}$, then we have $W_i \nsubseteq \mathrm{Im}(A)$ for some $i \in [n]$. For our purposes, we need to quantify when exactly this happens. Therefore we define:

Definition 4.3 (Wong Index). Let $\mathcal{B} \leq \mathbb{F}^{n\times n}$ be a matrix space, $A \in \mathcal{B}$ and $(W_i)_{i\in[[n]]}$ be the second Wong sequence of $(A, \mathcal{B})$. Let $k \in [[n]]$ be the maximum integer such that $W_k \subseteq \mathrm{Im}(A)$. Then k is called the *Wong index* of $(A, \mathcal{B})$. We shall denote it by $w(A, \mathcal{B})$.

Using the above definition, another way to state Lemma 4.9 is that if the Wong index $w(A, \mathcal{B})$ of $(A, \mathcal{B})$ is n, then A is of maximum rank in $\mathcal{B}$. But can one say more? In next section, we explore this connection. Consider a matrix space $\langle A, B\rangle$ generated by two matrices A, B. We shall prove that the closer $w(A, \langle A, B\rangle)$ is to n, the closer the rank of A is to the commutative rank of $\langle A, B\rangle$.

The converse of Lemma 4.9 is not true in general. But the converse is true in the special case when $\mathcal{B}$ is spanned by just two matrices. Fortunately, for the analysis of our algorithm we only require the converse to be true in this special case. The following fact from [Iva+15] formally states this idea.

Fact 4.3 (Restatement of Fact 11 in [Iva+15]). *Assume that $|\mathbb{F}| > n$ and let $A, B \in \mathbb{F}^{n\times n}$. If A is of maximum rank in $\langle A, B\rangle$, then the Wong index $w(A, \langle A, B\rangle)$ of $(A, \langle A, B\rangle)$ is n.*

Now we establish a connection between the commutative rank and Wong index in Section 4.5.

4.5 Relation between rank and Wong index

We prove that the natural greedy strategy of Algorithm 4.4 works, essentially by showing that either of the following happens:

1. The Wong index of the matrix obtained by Algorithm 4.4 at a given step is high enough, in which case, we show that the matrix already has the desired rank. Lemma 4.12 formalizes this.

2. We can increase the rank by a greedy step. Lemma 4.14 formalizes this.

Notice that this is similar to the approach described at the beginning of Section 4.3. Here Wong index is analogous to the degree of non-zero monomials of $\det(C)$.

In the above spirit, we quantify the connection between the commutative rank and Wong index in this section, using a series of lemmas. First we need the following lemma which demonstrates that the second Wong sequence remains "almost" the same under invertible linear maps.

Lemma 4.10. *Let $\mathbb{F}$ be any field, $A \in \mathcal{B} \leq \mathbb{F}^{n\times n}$ and $(W_i)_{i\in[[n]]}$ be the second Wong sequence of $(A, \mathcal{B})$. If $P \in \mathbb{F}^{n\times n}$ and $Q \in \mathbb{F}^{n\times n}$ are invertible matrices, then the second Wong sequence of $(PAQ, P\mathcal{B}Q)$ is $(PW_i)_{i\in[[n]]}$. In particular, $w(A, \mathcal{B}) = w(PAQ, P\mathcal{B}Q)$.*

Proof. Consider the i^{th} entry W'_i in the second Wong sequence of $(PAQ, P\mathcal{B}Q)$. We prove that $W'_i = PW_i$ for all $i \in [[n]]$. We use induction on i. The statement is trivially true for $i = 0$. By the induction hypothesis, we have, $W'_i = P\mathcal{B}Q(PAQ)^{-1}PW_{i-1} = P\mathcal{B}QQ^{-1}A^{-1}P^{-1}PW_{i-1} = P\mathcal{B}A^{-1}(W_{i-1}) = PW_i$. □

The following technical Lemma 4.11 relates the Wong index with a sequence of vanishing matrix products. Note the similarity of Lemma 4.11 to Lemma 4.5 and Lemma 4.6

Lemma 4.11. *Let $\mathbb{F}$ be any field and $A, B \in \mathbb{F}^{n\times n}$. Assume $A = \begin{bmatrix} I_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$ and express the matrix B as*

$$B = \begin{array}{c} \overbrace{\phantom{r \text{ columns}}}^{r \text{ columns}} \\ r \text{ rows } \{ \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \} n - r \text{ rows} \\ \underbrace{\phantom{n - r \text{ columns}}}_{n-r \text{ columns}} \end{array} \tag{4.14}$$

Let $\ell \leq n$ be the maximum integer such that first ℓ elements of the sequence of matrices

$$B_{22}, B_{21}B_{12}, B_{21}B_{11}B_{12}, \ldots, B_{21}B_{11}^{i}B_{12}, \ldots \tag{4.15}$$

are equal to the zero matrix. Then $\ell = w(A, \langle A, B\rangle)$.

Proof. Let $k = w(A, \langle A, B\rangle)$, we want to show that $k = \ell$. Notice that I_n is a pseudo-inverse of A, so we can assume $A' = I_n$ in the statement of Lemma 4.8. Consider the second Wong sequence of $(A, \langle A, B\rangle)$. By Lemma 4.8, it equals $(\langle A, B\rangle A')^i(\mathrm{Ker}(AA'))$ for $i \leq k$. Since we can assume that $A' = I_n$, this sequence is $(\langle A, B\rangle)^i(\mathrm{Ker}(A))$. $\mathrm{Ker}(A) \leq \mathbb{F}^n$ contains exactly the vectors having the first r entries equal to zero and $\mathrm{Im}(A)$ contains exactly the vectors which have the last $n - r$ entries equal to zero.

First we show that $\ell \geq k$. For this, we need to show that $B_{22} = B_{21}B_{12} = B_{21}B_{11}B_{12} = \ldots = B_{21}B_{11}^{k-2}B_{12} = \mathbf{0}$. If $k = 0$ then we do not need to show anything. Otherwise $k > 0$.

Consider the first entry W_1 of second Wong sequence of $(A, \langle A, B\rangle)$. By Lemma 4.8, we know that $W_1 = \langle A, B\rangle \operatorname{Ker}(A)$. As $\operatorname{Ker}(A) \leq \mathbb{F}^n$ contains exactly the vectors which have first r entries to be zero, if B_{22} was not zero then $B \operatorname{Ker}(A)$ would contain a vector with a non-zero entry in the last $n-r$ coordinates. This would violate the assumption $W_1 \subseteq \operatorname{Im}(A)$. Thus $B_{22} = \mathbf{0}$. Now we use induction on length of the sequence $B_{22}, B_{21}B_{12}, B_{21}B_{11}B_{12}, \ldots, B_{21}B_{11}^{i}B_{12}$. Our induction hypothesis assumes that for $i \geq 1$

$$B^i = \begin{matrix} \\ r \text{ rows} \end{matrix} \overbrace{\begin{matrix} \\ \left\{ \begin{bmatrix} \overbrace{B_{11}^i + \sum_{j=0}^{i-2} B_{11}^j B_{12} B_{21} B_{11}^{i-2-j}}^{r \text{ columns}} & \underbrace{B_{11}^{i-1}B_{12}}_{n-r \text{ columns}} \\ B_{21}B_{11}^{i-1} & \mathbf{0} \end{bmatrix} \right\} n-r \text{ rows} \end{matrix}} \tag{4.16}$$

and $B_{22} = B_{21}B_{12} = B_{21}B_{11}B_{12} = \ldots = B_{21}B_{11}^{i-2}B_{12} = \mathbf{0}$. We just proved the base case of $i = 1$. Consider the following evaluation of $B^{i+1} = B \cdot B^i$

$$B^{i+1} = \begin{matrix} \\ r \text{ rows} \end{matrix} \overbrace{\begin{matrix} \\ \left\{ \begin{bmatrix} \overbrace{B_{11}^{i+1} + \sum_{j=0}^{i-2} B_{11}^{j+1} B_{12} B_{21} B_{11}^{i-2-j} + B_{12}B_{21}B_{11}^{i-1}}^{r \text{ columns}} & B_{11}^i B_{12} \\ B_{21}B_{11}^i + \sum_{j=0}^{i-2} B_{21} B_{11}^j B_{12} B_{21} B_{11}^{i-2-j} & \underbrace{B_{21}B_{11}^{i-1}B_{12}}_{n-r \text{ columns}} \end{bmatrix} \right\} n-r \text{ rows} \end{matrix}} \tag{4.17}$$

Since $i+1 \leq k$, we must have $B_{21}B_{11}^{i-1}B_{12} = \mathbf{0}$, otherwise we would have $W_{i+1} \not\subseteq \operatorname{Im}(A)$. Also we know by the induction hypothesis that $B_{22} = B_{21}B_{12} = B_{21}B_{11}B_{12} = \ldots = B_{21}B_{11}^{i-2}B_{12} = \mathbf{0}$, this implies that

$$B^{i+1} = B \cdot B^i = \begin{matrix} \\ r \text{ rows} \end{matrix} \overbrace{\begin{matrix} \\ \left\{ \begin{bmatrix} \overbrace{B_{11}^{i+1} + \sum_{j=0}^{i-1} B_{11}^j B_{12} B_{21} B_{11}^{i-1-j}}^{r \text{ columns}} & \underbrace{B_{11}^i B_{12}}_{n-r \text{ columns}} \\ B_{21}B_{11}^i & \mathbf{0} \end{bmatrix} \right\} n-r \text{ rows} \end{matrix}} \tag{4.18}$$

Now we show that $k \geq \ell$. Since $k = w(A, \langle A, B\rangle)$, for all $1 \leq i \leq k$, B^i can be written as

$$B^i = \begin{array}{c} \\ r \text{ rows} \\ \\ \\ \end{array} \overbrace{\left\{\begin{array}{|cc|}\hline B_{11}^i + \sum_{j=0}^{i-2} B_{11}^j B_{12} B_{21} B_{11}^{i-2-j} & B_{11}^{i-1} B_{12} \\ B_{21} B_{11}^{i-1} & \underbrace{\mathbf{0}}_{n-r \text{ columns}} \\ \hline \end{array}\right\}}^{r \text{ columns}} n-r \text{ rows} \tag{4.19}$$

Note that $\langle A, B\rangle^i$ is spanned by all matrices of the form $M_1 M_2 \cdots M_i$ with $M_j = A$ or $M_j = B$, $1 \leq j \leq i$. Since we have that $W_k \subseteq \mathrm{Im}(A)$, we know $M_1 M_2 \cdots M_k \ker(A) \subseteq \mathrm{Im}(A)$ holds for any product $M_1 M_2 \cdots M_k$ as above. Now let us see what condition one needs such that $W_{k+1} \not\subseteq \mathrm{Im}(A)$ is true. Since A is the identity on $\mathrm{Im}(A)$, only B^{k+1} can take $\ker(A)$ out of $\mathrm{Im}(A)$ for $W_{k+1} \not\subseteq \mathrm{Im}(A)$ to be true. By a similar argument as above, this happens only when $B_{21} B_{11}^{k-1} B_{12} \neq \mathbf{0}$, thus $\ell \leq k$. □

Now, having established the connection between the Wong index and the sequence of vanishing matrix products, we prove another technical lemma establishing the relation between the length of this sequence and the commutative rank.

Finally, combining the above three lemmas, the following lemma gives the desired quantitative relation between the commutative rank and Wong index, essential to an alternative analysis of our algorithm. It shows that higher the Wong index of the given matrix, the better it approximates the rank of the space.

Lemma 4.12. *Let $\mathbb{F}$ be any field, $A \in \mathcal{B} = \langle B_1, B_2, \ldots, B_m\rangle \leq \mathbb{F}^{n\times n}$ and $B = \sum_{i=1}^m x_i B_i$, then*

$$\mathrm{crk}(\mathcal{B}) = \mathrm{crk}(\langle A, B\rangle) \leq \mathrm{rank}(A)\left(1 + \frac{1}{w(A, \langle A, B\rangle)}\right). \tag{4.20}$$

Proof. Let $\mathrm{rank}(A) = r$. We use $\mathcal{C}$ to denote the matrix space $\langle A, B\rangle$, note that this space is being considered over the rational function field $\mathbb{F}(x_1, x_2, \ldots, x_m)$.

We know that there exist matrices $P, Q \in \mathbb{F}^{n\times n}$ such that

$$PAQ = \begin{bmatrix} I_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{4.21}$$

Notice that $\mathrm{Im}(PAQ) = P\,\mathrm{Im}(A)$. Thus by Lemma 4.10, $w(A, \mathcal{C}) = w(PAQ, P\mathcal{C}Q)$. Also, it follows from Fact 3.1 that $\mathrm{rank}(A) = \mathrm{rank}(PAQ)$ and $\mathrm{crk}(\mathcal{C}) = \mathrm{crk}(P\mathcal{C}Q)$. Hence it is enough to show that

$$\mathrm{crk}(P\mathcal{C}Q) \leq \mathrm{rank}(PAQ)\left(1 + \frac{1}{w(PAQ, P\mathcal{C}Q)}\right). \tag{4.22}$$

For the sake of simplicity, we just write $P\mathcal{C}Q$ as $\mathcal{C}$ and PAQ as A. Thus we have

$$A = \begin{bmatrix} I_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{4.23}$$

We write B as

$$B = \begin{array}{c} \overbrace{\phantom{r \text{ columns}}}^{r \text{ columns}} \\ r \text{ rows } \{ \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \} n-r \text{ rows} \\ \underbrace{\phantom{n-r \text{ columns}}}_{n-r \text{ columns}} \end{array} \tag{4.24}$$

We get that B_{11} is non-singular over the field $\mathbb{F}(x_1, x_2, \ldots, x_m)$ since $A \in \mathcal{B}$. Also, we get by Lemma 4.11 that the first $w(A, \mathcal{C})$ entries of the sequence of matrices $B_{22}, B_{21}B_{12}, B_{21}B_{11}B_{12}, \ldots, B_{21}B_{11}^{i}B_{12} \ldots$ are zero matrices. Now we apply Lemma 4.6 to obtain that

$$\operatorname{rank}(B) = \operatorname{crk}(\mathcal{B}) = \operatorname{crk}(\mathcal{C}) \leq \operatorname{rank}(A) \left(1 + \frac{1}{w(A, \mathcal{C})}\right). \tag{4.25}$$

□

Lemma 4.12 essentially states that for the matrix spaces $\mathcal{C}$ which are spanned by two matrices, if the Wong index $w(A, \mathcal{C})$ for any matrix $A \in \mathcal{C}$ is "large" then $\operatorname{rank}(A)$ is close to $\operatorname{crk}(\mathcal{C})$. By using Theorem 3.3 and Lemma 4.12 we see that if $\mathcal{C}$ is generated by two matrices then for any matrix $A \in \mathcal{C}$, the following inequality holds.

$$\operatorname{ncrk}(\mathcal{C}) = \operatorname{crk}(\mathcal{C}) \leq \operatorname{rank}(A) \left(1 + \frac{1}{w(A, \mathcal{C})}\right). \tag{4.26}$$

Now it it very natural ask whether Equation (4.26) also holds for general matrix spaces? We prove that it does. Note that this does not help us in getting any algorithm exactly but it strengthens Lemma 4.12.

Lemma 4.13. *Let $\mathbb{F}$ be any field, $A \in \mathcal{B} = \langle B_1, B_2, \ldots, B_m \rangle \leq \mathbb{F}^{n \times n}$. Then we have:*

$$\operatorname{ncrk}(\mathcal{B}) \leq \operatorname{rank}(A) \left(1 + \frac{1}{w(A, \mathcal{B})}\right). \tag{4.27}$$

Proof. Let $\operatorname{rank}(A) = r$. We know that there exist matrices $P, Q \in \mathbb{F}^{n \times n}$ such that:

$$PAQ = \begin{bmatrix} I_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{4.28}$$

By using Lemma 4.10, we know that $w(A, \mathcal{B}) = w(PAQ, P\mathcal{B}Q)$. By Fact 3.1, we know that $\text{ncrk}(\mathcal{B}) = \text{ncrk}(P\mathcal{B}Q)$. Hence it is enough to show that:

$$\text{ncrk}(P\mathcal{B}Q) \leq \text{rank}(PAQ)\left(1 + \frac{1}{w(PAQ, P\mathcal{B}Q)}\right). \tag{4.29}$$

For the sake of simplicity, we again write $P\mathcal{B}Q$ as $\mathcal{B}$ and PAQ as A. Thus we can assume that:

$$A = \begin{bmatrix} I_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{4.30}$$

Fix an arbitrary $d \in \mathbb{N}^+$. Now consider the following matrix $A^{[d]} \in \mathcal{B}^{[d]}$ defined as:

$$A^{[d]} \stackrel{\text{def}}{=\!=} I_d \otimes A$$

Consider the symbolic matrix $B^{[d]} \stackrel{\text{def}}{=\!=} X_1 \otimes B_1 + \cdots + X_m \otimes B_m$ in variables $x^i_{j,k}$ for $i \in [m]$ and $j, k \in [d]$. Here the matrix X_i is composed of variables $x^i_{j,k}$. First we want to show that $w(A, \mathcal{B}) \leq w(A^{[d]}, \langle A^{[d]}, B^{[d]} \rangle)$. Now observe that I_{nd} is a pseudo inverse of $A^{[d]}$. Suppose $k = w(A, \mathcal{B})$. Now we observe the following facts.

1. $A' \stackrel{\text{def}}{=\!=} I_n$ is a pseudo inverse of A and thus $(\mathcal{B}A')^j = (\mathcal{B})^j$ for all $j \in \mathbb{N}$.
2. $(\mathcal{B})^j$ is spanned by the matrices $\{B_{i_1} \cdot B_{i_2} \cdot \cdots \cdot B_{i_j} \mid 1 \leq i_1, i_2, \ldots, i_j \leq m\}$.

By using Lemma 4.9 and the above facts, we conclude the following:

$$\forall t \leq k, \forall i_1, i_2, \ldots, i_t \in [m] : B_{i_1} \cdot B_{i_2} \cdot \cdots \cdot B_{i_t} \cdot \text{Ker}(A) \subseteq \text{Im}(A). \tag{4.31}$$

Now we observe the following equality:

$$(B^{[d]})^j = \sum_{1 \leq i_1, i_2, \ldots, i_j \leq m} (X_{i_1} \cdot X_{i_2} \cdot \cdots \cdot X_{i_j}) \otimes (B_{i_1} \cdot B_{i_2} \cdot \cdots \cdot B_{i_j}) \tag{4.32}$$

Equation (4.31) and Equation (4.32) imply that $w(A^{[d]}, \langle A^{[d]}, B^{[d]} \rangle) \geq k$. Now we apply Lemma 4.12 to $A^{[d]}$ and $B^{[d]}$ to obtain the following inequality:

$$\text{crk}(\mathcal{B}^{[d]}) = \text{rank}(B^{[d]}) = \text{crk}(\langle A^{[d]}, B^{[d]} \rangle) \leq \text{rank}(A^{[d]})\left(1 + \frac{1}{w(A^{[d]}, \langle A^{[d]}, B^{[d]} \rangle)}\right). \tag{4.33}$$

The first equality in Equation (4.33) follows from Lemma 3.12. Thus Equation (4.33) implies $\text{crk}(\mathcal{B}^{[d]}) \leq rd\left(1 + \frac{1}{k}\right)$. Now we choose d to be the positive integer which achieves the maximum in Theorem 3.2. This implies that:

$$\text{ncrk}(\mathcal{B}) = \frac{\text{crk}(\mathcal{B}^{[d]})}{d} \leq r\left(1 + \frac{1}{k}\right).$$

□

Lemma 4.14. *If* $|\mathbb{F}| > n$, $A \in \mathcal{B} = \langle B_1, B_2, \ldots, B_m \rangle \leq \mathbb{F}^{n\times n}$, $B = \sum_{i=1}^{m} x_i B_i$ *and* $w(A, \langle A, B\rangle) < k$ *for some* $k \in [n]$, *then there exist* $1 \leq i_1, i_2, \ldots, i_k \leq m$ *and* $\lambda_1, \lambda_2, \ldots, \lambda_k \in \mathbb{F}$ *such that* $w(A, \langle A, C\rangle) < k$, *where* $C = \lambda_1 B_{i_1} + \lambda_2 B_{i_2} + \ldots + \lambda_k B_{i_k}$.

Proof. Let $\text{rank}(A) = r$. We know that there exist matrices $P, Q \in \mathbb{F}^{n\times n}$ such that

$$PAQ = \begin{bmatrix} I_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{4.34}$$

Let $A' = PAQ$, $\mathcal{B}' = P\mathcal{B}Q$ and $B' = \sum_{i=1}^{m} x_i PB_iQ$. We write B' as

$$B' = \begin{array}{c} \overbrace{}^{r \text{ columns}} \quad \overbrace{}^{n-r \text{ columns}} \\ \left.\begin{bmatrix} B'_{11} & B'_{12} \\ B'_{21} & B'_{22} \end{bmatrix}\right\} \begin{array}{l} r \text{ rows} \\ n-r \text{ rows} \end{array} \end{array} \tag{4.35}$$

By using Lemma 4.10, we know that $w(A, \langle A, B\rangle) = w(A', \langle A', B'\rangle) < k$. By using Lemma 4.11 we get that there exists $t \leq k$ such that $B'_{21}(B'_{11})^{t-2}B'_{12} \neq \mathbf{0}$ and

$$(B')^t = \begin{array}{c} \overbrace{}^{r \text{ columns}} \quad \overbrace{}^{n-r \text{ columns}} \\ \left.\begin{bmatrix} B''_{11} & B''_{12} \\ B''_{21} & B'_{21}(B'_{11})^{t-2}B'_{12} \end{bmatrix}\right\} \begin{array}{l} r \text{ rows} \\ n-r \text{ rows} \end{array} \end{array} \tag{4.36}$$

for some matrices $B''_{11}, B''_{12}, B''_{21}$. Since the entries of the matrix $B'_{21}(B'_{11})^{t-2}B'_{12}$ are polynomials in the variables $x_1, x_2, \ldots, x_m$ of degree at most $t \leq k$, there exists an assignment to these variables by field constants, assigning at most k variables non-zero values such that $B'_{21}(B'_{11})^{t-2}B'_{12}$ evaluates to a non-zero matrix. By using Lemma 4.11 again, this assignment gives us a matrix $C' \in \mathcal{B}'$ such that $w(A', \langle A', C'\rangle) < k$. By using Lemma 4.10, the same assignment of the variables gives us a matrix $C \in \mathcal{B}$ such that $w(A, \langle A, C\rangle) < k$. □

4.6 An Alternative proof of correctness of Algorithm 4.4

Here we give an alternative proof of correctness of Algorithm 4.4. This proof will be analogous to the proof of Theorem 4.5. The minimum degree of non-zero monomials in $\det(C)$ in Theorem 4.5 is analogous to the Wong index of A in the ensuing discussion.

We have a matrix space $\mathcal{B} = \langle B_1, B_2, \ldots, B_m \rangle \leq \mathbb{F}^{n \times n}$, $B = \sum_{i=1}^{m} x_i B_i$ and a matrix $A \in \mathcal{B}$. Our goal is find a matrix D in $\mathcal{B}$ such that its rank is "close" to the commutative rank of $\mathcal{B}$. If the Wong index $w(A, \langle A, B \rangle)$ of A in $\langle A, B \rangle$ is "large", then we know by Lemma 4.12 that rank of of A is "close" to the commutative rank of $\mathcal{B}$, which is equal to the commutative rank of $\langle A, B \rangle$. What if this Wong index $w(A, \langle A, B \rangle)$ is "small"? Then we know by Lemma 4.14 that by trying out a small number (that means, $m^{w(A,\mathcal{B})+1}$) of possibilities of combinations of B_i, we can find a matrix $C \in \mathcal{B}$ such that Wong index $w(A, \langle A, C \rangle)$ of A in $\langle A, C \rangle$ is also "small". Using Fact 4.3 we obtain that rank of A is not maximum in $\langle A, C \rangle$. Thus there exists $\lambda \in \mathbb{F}$ such that $\operatorname{rank}(A + \lambda C) > \operatorname{rank}(A)$. And we can find this λ quite efficiently. Also, $A + \lambda C \in \mathcal{B}$. Thus we can efficiently find a matrix of bigger rank if we are given a matrix of "small" Wong index. This idea essentially implies that Algorithm 4.4 is a PTAS for computing the commutative rank.

The following Theorem 4.6 gives a different proof of the correctness of Algorithm 4.4. Let s be an upper bound on the bit size of the entries of $B_1, \ldots, B_m$.

Theorem 4.6. *Assume that $|\mathbb{F}| > n$. Algorithm 4.4 runs in time $O((mn)^{\frac{1}{\epsilon}} \cdot M(n, s + \log n) \cdot n)$ and returns a matrix $A \in \mathcal{B}$ such that $\operatorname{rank}(A) \geq (1 - \epsilon) \cdot \operatorname{crk}(\mathcal{B})$, where $M(n, t)$ is the time required to compute the rank of an $n \times n$ matrix with entries of bit size at most t.*

Proof. Suppose $B = \sum_{i=1}^{m} x_i B_i$ and A be the rank r matrix returned by Algorithm 4.4. Let k be the Wong index $w(A, \langle A, B \rangle)$ of $(A, \langle A, B \rangle)$. By Lemma 4.12 we know that $\operatorname{crk}(\mathcal{B}) \leq r\left(1 + \frac{1}{k}\right)$. Thus $r \geq (1 - \frac{1}{k+1}) \operatorname{crk}(\mathcal{B})$. If $\epsilon \geq \frac{1}{k+1}$, then we are done. Otherwise we have that $\epsilon < \frac{1}{k+1}$, *i.e.*, $k < \frac{1}{\epsilon} - 1$. Since $\ell = \lceil \frac{1}{\epsilon} - 1 \rceil$, we also have $w(A, \langle A, B \rangle) < \ell$. By using Lemma 4.14, we get that there exist $1 \leq i_1, i_2, \ldots, i_\ell \leq m$ and $\lambda_1, \lambda_2, \ldots, \lambda_m \in \mathbb{F}$ such that that $w(A, \langle A, C \rangle) < \ell$, where $C = \lambda_1 B_{i_1} + \lambda_2 B_{i_2} + \ldots + \lambda_\ell B_{i_\ell}$. By using Fact 4.3, we get that A is not of maximum rank in $\langle A, C \rangle$. Thus there exists $\lambda \in \mathbb{F}$ such that $\operatorname{rank}(A + \lambda C) > \operatorname{rank}(A)$, and we shall detect this in Algorithm 4.4, since we try all possible choices of $i_1, i_2, \ldots, i_\ell$.

The running time proof is exactly the same as in the proof of Theorem 4.5. □

4.7 Tight examples

We conclude by giving some tight examples, which show that the analysis of the approximation performance of Algorithm 4.4 cannot be improved. Consider the following matrix space of $n \times n$-matrices:

$$\left(\begin{array}{cccc|cccc} * & 0 & \dots & 0 & * & 0 & \dots & 0 \\ 0 & * & \dots & 0 & 0 & * & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & * & 0 & 0 & \dots & * \\ \hline 0 & 0 & \dots & 0 & * & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & * & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & * \end{array}\right) \tag{4.37}$$

Each block has size $\frac{n}{2} \times \frac{n}{2}$. This space consists of all matrices where we can substitute arbitrary values for the $*$ and the basis consists of all matrices where exactly one $*$ is replaced by 1 and all others are set to 0. Assume that $\epsilon = \frac{1}{2}$, that means, that the greedy algorithm only looks at sets of size $\ell = 1$. Furthermore, assume that the matrix A constructed so far is:

$$A = \begin{pmatrix} \mathbf{0} & I_{\frac{n}{2}} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}. \tag{4.38}$$

Any single basis matrix cannot improve the rank of A, since either its non-zero column is contained in the column span of A or its non-zero row is contained in the row span

of A. On the other hand, the matrix space contains a matrix of full rank n, namely, the identity matrix. The next space for the case $\ell = 2$ looks like this:

$$\left(\begin{array}{cccc|cccc|cccc} * & 0 & \dots & 0 & * & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & * & \dots & 0 & 0 & * & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & * & 0 & 0 & \dots & * & 0 & 0 & \dots & 0 \\ \hline 0 & 0 & \dots & 0 & * & 0 & \dots & 0 & * & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & * & \dots & 0 & 0 & * & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & * & 0 & 0 & \dots & * \\ \hline 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & * & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & * & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & * \end{array}\right) \tag{4.39}$$

and the corresponding matrix A is

$$A = \begin{pmatrix} \mathbf{0} & I_{\frac{2n}{3}} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}. \tag{4.40}$$

By an argument similar to above, it is easy to see that we need at least three matrices to improve the rank of A, so the algorithm gets stuck with a $\frac{2}{3}$-approximation. The above scheme generalizes to arbitrary values of ℓ in the obvious way.

Part II

REAL ROOT COMPUTATION OF SPARSE POLYNOMIALS

5

COMPUTING THE ROOTS OF POLYNOMIALS

Computing the roots of a uni-variate polynomial is an important problem in theory and practice. For instance, the roots of the characteristic polynomial of a matrix A are exactly the eigen-values of A. In general, many computational tasks from mathematics, engineering and computer science reduce to solving a system of polynomial equations. It is well known that solving a system of polynomial equations can be reduced to solving a polynomial equation in one variable. For example, the authors in [BS16] achieve this by using the concept of *separating linear forms*. There are many other methods known to solve a system of polynomial equations, but all of them reduce to uni-variate root finding. Therefore, the problem of computing the roots of a uni-variate polynomial has been studied extensively in the literature.

In this chapter, we consider only the real polynomials. In a lot of applications, one wants to only compute all the real roots of a real polynomial. In addition, in some applications, one only wants to compute the real roots of a real sparse polynomial, *i.e.*, only few monomials have non-zero coefficients.

It might happen that a given real uni-variate polynomial has very few real roots but the fundamental theorem of algebra states that any complex polynomial (in particular a real polynomial) of degree n has exactly n complex roots, *i.e.*, the field $\mathbb{C}$ of complex numbers is algebraically closed, a result that was already proved by Gauss in 1816. We give a folklore algebraic proof of the fundamental theorem of algebra in Section 5.1. In contrast, a real sparse polynomials can not have too many real roots (see Theorem 5.3).

This chapter can be read as follows. Section 5.1 gives an algebraic proof of the fundamental theorem of algebra, and we provide bounds on the magnitude of roots of complex polynomials (Cauchy's root bound). Section 5.2 introduces the various definitions and notations which are commonly used in this chapter. Section 5.3 focuses on the structure of real roots of real polynomials. Here we prove the well known Descartes's rule of signs. We also introduce the notions of Obreshkoff lens. This helps us to prove that the cone C_n (Figure 5.2) contains at most $k-1$ roots of any (n, k, τ)-nomial (Definition 5.2). This observation is crucial in the analysis of the main contribution of this chapter. Section 5.4 proves that the roots of trinomials are well separated. Using our algorithm (Theorem 5.13) to compute a strong covering, results of Section 5.4 imply that the real roots of trinomials can be computed in polynomial time.

Next we prove that the root separation result of trinomials can not be generalized to 4-nomials. More specifically, Section 5.5 demonstrates that the roots of the well known Mignotte like polynomials are not well separated. Moreover, we also show that any algorithm which isolates the real roots of such polynomials, is of exponential complexity (Theorem 5.10).

Thereafter, we give a very brief survey of the history of polynomial root computation algorithms. Section 5.7 introduces the notion of fractional derivatives, which is a crucial ingredient of our algorithm. We use this notion of fractional derivatives to expound a classical result of [CKS99], which states that the integer roots of integer (n, k, τ)-nomials can be computed in polynomial time.

Next we introduce the notion of weak and strong coverings. To give some intuition, we first give a brief description of algorithm claimed in Theorem 5.13, omitting the technical details. Section 5.10 contains several key results on polynomial arithmetic concerning the value of (n, k, τ)-nomials at admissible points and also computing the sign of an (n, k, τ)-nomial at a given point.

We also describe a refinement routine, which refines a isolating interval to a desire length. This refinement routine (Algorithm 5.11) is a key component of our algorithm to compute a weak covering.

Afterwards, we describe an algorithm (Algorithm 5.12) to compute a weak covering of any (n, k, τ)-nomial. Section 5.13 describes the so called $\tilde{T}_l$-test, which essentially counts the number of roots of a (n, k, τ)-nomial F in a given disk Δ. The novel contribution of Section 5.13 is that to perform the $\tilde{T}_l$-test, we only need to be concerned with first k^2 coefficients of the polynomial F_Δ. Finally, we combine the $\tilde{T}_l$-test and Algorithm 5.12 to prove our main result of this chapter (Theorem 5.13).

5.1 Complex Roots of Complex Polynomials

Lemma 5.1. *Let $f(x) \in \mathbb{R}[x]$ be a polynomial such that $f(a)f(b) < 0$ for two real numbers a and b where $a < b$, then there exists a root x_0 of $f(x)$ in interval (a, b), i.e., there exists a real number x_0 such that $f(x_0) = 0$ and $a < x_0 < b$.*

Proof. This directly follows from Rolle's Theorem. □

Corollary 5.1. *Let $f(x) \in \mathbb{R}[x]$ be a polynomial of odd degree then $f(x)$ has a real root.*

Proof. This follows by applying Lemma 5.1 on $a = -\infty$ and $b = +\infty$. □

We use e_i to denote the i^{th} elementary symmetric polynomial e_n^i in n variables $x_1, x_2, \ldots, x_n$ (see Chapter 6).

Lemma 5.2 ([DF04])**.** *Let K be any field and $f(x) \in K[x]$ be a monic polynomial of degree n, then there exists a field extension $L \mid K$ such that in L, $f(x)$ splits in linear factors. More specifically, $f(x) = \prod_{i=1}^{n}(x - z_i)$ with $z_i \in L$.*

Note that the z_i's in Lemma 5.2 may not be necessarily distinct. L is also called the *splitting field* of $f(x)$ over K.

Lemma 5.3. *Let $g(x) \in \mathbb{R}[x]$ be any non-constant real monic polynomial, then there exists a $z \in \mathbb{C}$ such that $g(z) = 0$.*

Proof. Let $n = \deg(g(x))$. We factorize n in the form $n = 2^k \cdot q$, where q is an odd number. We now prove the Lemma by induction on k. The base case $k = 0$ follows from Corollary 5.1. For the induction step, assume that $k \geq 1$. By using Lemma 5.2, we know that there exists a field extension $L \mid \mathbb{R}$ such that $g(x) = \prod_{i=1}^{n}(x - z_i)$ with $z_i \in L$. We want to prove that there exists an $i \in [n]$ such that $z_i \in \mathbb{C}$, in fact we shall prove that there exist i, j with $i \neq j$ such that $z_i, z_j \in \mathbb{C}$. Fix a real number $c \in \mathbb{R}$. For $\{i, j\} \in \binom{[n]}{2}$, we define:

$$w_{ij} \stackrel{\text{def}}{=\!=} z_i + z_j + cz_iz_j.$$

Note that $w_{ij} \in L$. Now let us define the polynomial $h(x) \in L[x]$ as below.

$$h(x) \stackrel{\text{def}}{=\!=} \prod_{\{i,j\} \in \binom{[n]}{2}} (x - w_{ij}).$$

Note that the coefficients of $h(x)$ are elementary symmetric polynomial in w_{ij}'s, hence these coefficients are real symmetric polynomials in z_i's as well. By using Theorem 6.1, we know that these coefficients are real polynomial functions of the n elementary symmetric polynomials $e_1, e_2, \ldots, e_n$ of $z_1, z_2, \ldots, z_n$. But the elementary symmetric polynomials in variables $z_1, z_2, \ldots, z_n$ are coefficients of $g(x)$ and hence they are real. This implies that coefficients of $h(x)$ are also real. Thus $h(x) \in \mathbb{R}[x]$. We have that $\deg(h(x)) = \binom{n}{2} = 2^{k-1} \cdot q(n-1)$. By using the fact that $k \geq 1$, we conclude that $q(n-1)$ is odd. Now we can apply induction hypothesis to conclude that there exists $\{i, j\}$ such that $w_{ij} \in \mathbb{C}$. Until now, we performed the above argument for a fixed $c \in \mathbb{R}$. Let $\{i, j\}$ be the lexicographically smallest pair such that $w_{ij} \in \mathbb{C}$. This defines a function $\tau : \mathbb{R} \to \binom{[n]}{2}$ as below.

$$\begin{aligned} \tau : \mathbb{R} &\to \binom{[n]}{2} \\ c &\mapsto \{i, j\} \text{ such that } \{i, j\} \text{ is the lexiographically smallest pair with } w_{ij} \in \mathbb{C} \end{aligned}$$

Since $\binom{[n]}{2}$ is a finite set and $\mathbb{R}$ is infinite, we get that there exists distinct real numbers c and c' such that $\tau(c) = \tau(c')$. Thus there exists a pair $\{i, j\}$ such that both $z_i + z_j + cz_iz_j$ and $z_i + z_j + c'z_iz_j$ are in $\mathbb{C}$. Hence there difference $(c - c')z_iz_j$ is also in $\mathbb{C}$. Since $(c - c')$ is a non-zero real number, we conclude that $z_iz_j \in \mathbb{C}$ as well. This implies that $z_i + z_j \in \mathbb{C}$ as well. Since z_i and z_j are the roots of quadratic equation $x^2 - (z_i + z_j)x + z_iz_j$ and square root of complex numbers are complex, we get that z_i and z_j are also complex numbers. □

Corollary 5.2. *Let $f(x) \in \mathbb{C}[x]$ be any non-constant polynomial, then there exists a $z \in \mathbb{C}$ such that $f(z) = 0$.*

Proof. We can assume $f(x)$ to be monic. Define $g(x)$ to be the polynomial $f(x) \cdot \overline{f}(x)$. Here $\overline{f}(x)$ is the complex conjugate of $f(x)$, *i.e.*, we obtain $\overline{f}(x)$ from $f(x)$ by taking the complex conjugate of each coefficient of $f(x)$. Note that $\overline{g}(x) = \overline{f(x)\overline{f}(x)} = \overline{f}(x)f(x) = g(x)$, hence $g(x)$ is a real polynomial. By using Lemma 5.3, we know that $g(x)$ has a complex root z. Therefore $g(z) = f(z) \cdot \overline{f}(z) = f(z) \cdot f(\overline{z}) = 0$. Thus either $f(z) = 0$ or $f(\overline{z}) = 0$. Thus z or $\overline{z}$ is a root of $f(x)$. □

Theorem 5.1 (Fundamental theorem of algebra)**.** *The field $\mathbb{C}$ of complex numbers is algebraically closed, i.e., any non-zero polynomial $f(x) \in \mathbb{C}[x]$ of degree n has exactly n complex roots.*

Proof. We prove it by induction on the degree n of $f(x) \in \mathbb{C}[x]$. The base case of $n = 1$ is trivially true. By using Corollary 5.2, we know that there there exists a $z \in \mathbb{C}$ such that $f(z) = 0$. By Little Bezout's Theorem (Factor Theorem) [Rud04] we know that $f(x) = (x - z)g(x)$, here $g(x) \in \mathbb{C}[x]$ is a polynomial of degree $n - 1$. By using the induction hypothesis, we know that $g(x)$ has exactly $n - 1$ complex roots. Therefore $f(x)$ has exactly n complex roots. □

For a more concise and analytical proof of Theorem 5.1, the reader is referred to Chapter 19 in [AZ09]. The magnitude of roots of a polynomial can also be bounded in terms of magnitude of coefficients using Cauchy's bound.

Theorem 5.2 (Cauchy root bound [Yapoo])**.** *Let $f(x) = \sum_{i=0}^{n} f_i x^i \in \mathbb{C}[x]$ be a complex polynomial of degree n. For every root α of f, the following inequality holds true.*

$$|\alpha| \leq 1 + \max\left(\left|\frac{f_0}{f_n}\right|, \left|\frac{f_1}{f_n}\right|, \ldots, \left|\frac{f_{n-1}}{f_n}\right|\right).$$

Proof. By way of contradiction, assume that there exists a root α of $f(x)$ such that $|\alpha| > 1 + \max\left(\left|\frac{f_0}{f_n}\right|, \left|\frac{f_1}{f_n}\right|, \ldots, \left|\frac{f_{n-1}}{f_n}\right|\right)$. Since $f(\alpha) = \sum_{i=0}^{n} f_i \alpha^i = 0$, we have that:

$$-\alpha^n = \sum_{i=0}^{n-1} \frac{f_i}{f_n} \alpha^i.$$

Therefore, we have:

$$\begin{aligned}
|-\alpha^n| = |\alpha|^n &= \left|\sum_{i=0}^{n-1} \frac{f_i}{f_n}\alpha^i\right| \\
&\leq \sum_{i=0}^{n-1} \left(\left|\frac{f_i}{f_n}\right| \cdot |\alpha|^i\right) \\
&\leq \max_i \left\{\left|\frac{f_i}{f_n}\right|\right\} \cdot \left(\sum_{i=0}^{n-1} |\alpha|^i\right) \\
&< (|\alpha| - 1) \cdot \left(\frac{|\alpha|^n - 1}{|\alpha| - 1}\right) = |\alpha|^n - 1. \qquad (5.1)
\end{aligned}$$

Equation (5.1) above is obviously a contradiction, hence any root α of $f(x)$ satisfies $|\alpha| \leq 1 + \max\left(\left|\frac{f_0}{f_n}\right|, \left|\frac{f_1}{f_n}\right|, \ldots, \left|\frac{f_{n-1}}{f_n}\right|\right)$. □

5.2 Definitions and Notations

1. For a complex polynomial $f(x) \in \mathbb{C}[x]$, the root separation $\sigma(f)$ of $f(x)$ is defined as the minimal distance between any two distinct roots of $f(x)$.
2. For a complex number c and a positive real number r, $\Delta_r(c)$ is used to denote the open disk in the complex plane with center c and radius r.
3. For a real interval $I = (a, b)$, $w(I)$ is used to denote the width $b - a$ of I.
4. For a real interval $I = (a, b)$, $m(I)$ is used to denote the center $\frac{a+b}{2}$ of I.
5. For a real interval $I = (a, b)$, we use $\Delta(I)$ to denote the once circle region of I. That is, $\Delta(I) \overset{\text{def}}{=\!=} \Delta_{\frac{w(I)}{2}}(m(I))$.
6. We use $\max_1(x_1, x_2, \ldots, x_m)$ to denote $\max(1, |x_1|, |x_2|, \ldots, |x_m|)$ for arbitrary $x_1, x_2, \ldots, x_m \in \mathbb{C}$, log is used to denote $\log_2$ (the binary logarithm) and $\overline{\log}(x_1, x_2, \ldots, x_m)$ is used to denote the quantity $\max_1(\lceil \log(\max_1(x_1, x_2, \ldots, x_m)) \rceil)$.
7. For a complex polynomial $F(x) = \sum_{i=0}^{n} A_i x^i$ with roots $\xi_1, \xi_2, \ldots, \xi_n$, we define:
   a) $\sigma_i \overset{\text{def}}{=\!=} \sigma(\xi_i, F) \overset{\text{def}}{=\!=} \min_{j \neq i} |\xi_i - \xi_j|$ the *separation* of the root ξ_i.
   b) $\sigma_F \overset{\text{def}}{=\!=} \min_i \sigma_i$.

Whenever we encounter degree n polynomials in this chapter, we always assume that $n > 1$.

5.3 Real Roots of (Sparse) Real Polynomials

We have seen that any complex polynomial (in particular a real polynomial) of degree n has exactly n complex roots. But the number of real roots of a real polynomial of degree n may be far less than n. For instance, it is known that a real "random"[1] polynomial of degree n has $\Theta(\log n)$ real roots in expectation[Kac43; EK95]. In a lot of real world applications, we want to find the (real) roots of a given (real) polynomial. And usually the given polynomial is "sparse". Here "sparse" means that only a very few coefficients of the polynomial are non-zero. In this case, it is reasonable to describe the polynomial by listing only the non-zero coefficients and the corresponding monomials. To this end, we define the notion of k-sparse polynomials or simply k-nomials.

Definition 5.1 (k-sparse or k-nomial)**.** A polynomial $f(x) \in \mathbb{C}[x]$ is said to be k-sparse or k-nomial if the number of non-zero coefficients in $f(x)$ is at most k, *i.e.*, $f(x)$ can be written as in Equation (5.2).

$$f(x) = \sum_{i=1}^{k} f_i x^{e_i} \text{ with } f_i \in \mathbb{C} \text{ and } 0 \leq e_1 < e_2 < \ldots < e_k \leq n, e_i \in \mathbb{N}. \tag{5.2}$$

In whatever follows, we are primarily concerned with sparse real polynomials and their roots. To study the computation of the real roots of real k-nomials, we define the notion of (n, k, τ)-nomials.

Definition 5.2 ((n, k, τ)-nomial)**.** A real polynomial $f(x) \in \mathbb{R}[x]$ is said to be an (n, k, τ)-nomial if it can be written as in Equation (5.3) below.

$$f(x) = \sum_{i=1}^{k} f_i x^{e_i}. \tag{5.3}$$

Here $0 \leq e_1 < e_2 < \cdots < e_k \leq n$ and $2^{-\tau} \leq |f_i| \leq 2^{\tau}$.

Notice that, for an integer (n, k, τ)-nomial $f(x)$, the sparse representation of f can be described using $O(k(\log n + \tau))$ bits.

If we want to "compute" the complex roots of $f(x)$ then we obviously need $\Omega(n)$ time to "compute" these roots as there are exactly n complex roots. Thus there is no hope for an algorithm which computes all the complex roots in time polynomial in the "input size" $O(k(\log n + \tau))$.

But Descartes's rule of sign (Theorem 5.3) implies that any (n, k, τ)-nomial has at most $k - 1$ positive real roots. This also implies that $f(x)$ has at most $k - 1$ negative real roots because the positive real roots of $f(-x)$ correspond to the negative real roots of $f(x)$. Thus, one can hope for an algorithm which "computes" all the real roots in

1 It has $\Theta(\log n)$ real roots in expectation when the coefficients are independent standard normals. If the coefficients f_i's are independent normals with variances $\binom{n}{i}$ then the number of expected real roots is $\sqrt{n}$.

time $\mathsf{poly}(k, \log n, \tau)$. For the sake of completeness, we prove Descartes's rule of sign here. We use the notation $\mathrm{var}(f)$ to denote the number of sign changes in the coefficient sequence of a real polynomial $f(x) \in \mathbb{R}[x]$. More specifically, if $f(x) = \sum_{i=1}^{k} f_i x^{e_i}$ with $e_1 < e_2 < \cdots < e_k$ then:

$$\mathrm{var}(f) \stackrel{\text{def}}{=\!=} \text{Number of signs changes in the sequence } (f_1, f_2, \ldots, f_k).$$

For example, we have:

$$\mathrm{var}(2 + 3x^2 - 5x^6 - 6x^{10} + 20x^{20}) = 2$$
$$\mathrm{var}(-1 - 13x^{10} + 5x^{11} + 7x^{16} + 2x^{30}) = 1$$

For a real polynomial $f(x) \in \mathbb{R}[x]$, we use the notation $N_+(f)$ to denote the number of positive real roots of f (counted with multiplicity).

Theorem 5.3 (Descartes's rule of signs, [Wano4; Eigo8])**.** *For any polynomial $f(x) \in \mathbb{R}[x]$, $\mathrm{var}(f) - N_+(f)$ is a non-negative even integer.*

Proof. Suppose $f(x) = \sum_{i=1}^{k} f_i x^{e_i}$ with $e_1 < e_2 < \cdots < e_k$. First note that we can assume $e_1 = 0$ because if $e_1 > 0$ then we can divide $f(x)$ by x^{e_1} and the positive real roots remain the same. Thus we assume $e_1 = 0$. Moreover, we assume that $f_1 > 0$ because if $f_1 < 0$ then we can just consider $-f(x)$. We first show that $\mathrm{var}(f) - N_+(f)$ is always even. We use the following observations for the multiplicities of the roots of polynomials.

- If $f(x)$ touches (but does not cross) the x-axis at some point $a \in \mathbb{R}^+$ then a is a root of $f(x)$ of even multiplicity.
- If $f(x)$ crosses the x-axis at some point $a \in \mathbb{R}^+$ then a is a root of $f(x)$ of odd multiplicity.

Now consider the following two cases.

Case 1. $f_k > 0$. In this case, $\mathrm{var}(f)$ is even because $f_1 > 0$. Also, we have $f(0) = f_1 > 0$ and $f(\infty) = \infty$. Using the above observations about the multiplicities of the roots of $f(x)$, we see that $N_+(f)$ is also even.

Case 2. $f_k < 0$. In this case, $\mathrm{var}(f)$ is odd because $f_1 > 0$. Also, we have $f(0) = f_1 > 0$ and $f(\infty) = -\infty$. Using the above observations about the multiplicities of the roots of $f(x)$, we see that $N_+(f)$ is also odd.

Thus $\mathrm{var}(f) - N_+(f)$ is always even. Now, we prove $N_+(f) \leq \mathrm{var}(f)$ by applying induction on the degree of $f(x)$. This inequality is trivially true for the degree one polynomials. Now consider the following cases.

Case 1. f_1 and f_2 are of same sign. In this case, $\text{var}(f) = \text{var}(f')$. By using Rolle's theorem, we know that $N_+(f') \geq N_+(f) - 1$. Therefore:

$$\begin{aligned} N_+(f) &\leq N_+(f') + 1 \\ &\leq \text{var}(f') + 1 \leq \text{var}(f) + 1. \end{aligned}$$

Since $\text{var}(f) - N_+(f)$ is even, we get that $N_+(f) < \text{var}(f) + 1$. Thus $N_+(f) \leq \text{var}(f)$.

Case 2. f_1 and f_2 are of different sign. In this case, $\text{var}(f') = \text{var}(f) - 1$. By again using Rolle's theorem, we obtain that:

$$\begin{aligned} N_+(f) &\leq N_+(f') + 1 \\ &\leq \text{var}(f') + 1 = \text{var}(f). \end{aligned}$$

□

Descartes's rule of signs (Theorem 5.3) also implies a bound on the number of all real roots of a real k-nomial.

Corollary 5.3. *For any real k-nomial $f(x) \in \mathbb{R}[x]$, the number of non-zero real roots (counted with multiplicity) of $f(x)$ is at most $2k - 2$.*

Proof. For any real k-nomial $f(x) \in \mathbb{R}[x]$, observe that $\text{var}(f) \leq k - 1$. Thus by using Descartes's rule of signs (Theorem 5.3), we obtain that number of positive real roots of $f(x)$ is at most $k - 1$. It is also easy to see that the negative real roots of $f(x)$ are in bijection with the positive real roots of $f(-x)$. Note that $f(-x)$ is also a real k-nomial. Thus the number of negative real roots of $f(x)$ is also at most $k - 1$. Hence the total number of non-zero real roots of $f(x)$ is at most $2k - 2$. □

In general, a k-nomial can have zero as a root with arbitrary multiplicity. Since we can easily check if zero is the root of a given polynomial, we are only concerned with non-zero roots. There are generalizations of Descartes's rule of signs which demonstrate that the roots of k-nomials have some additional geometry. To describe this geometry of roots of polynomials, we define:

Definition 5.3. Let $I = (a, b) \subseteq \mathbb{R}$ be an interval. For a real polynomial $f(x) \in \mathbb{R}[x]$ of degree n, we define :

$$f_I \stackrel{\text{def}}{=\!=} (x+1)^n f\left(\frac{ax+b}{x+1}\right)$$

$$\text{var}(f, I) \stackrel{\text{def}}{=\!=} \text{var}(f_I)$$

There is a one-to-one correspondence between the roots of f in the interval (a, b) and the positive real roots of via f_I the Möbius transformation that maps a point $x \in \mathbb{C} \setminus \{-1\}$ to $\frac{ax+b}{x+1} \in \mathbb{C}$. Therefore, $\text{var}(f_I)$ is an upper bound on the number of roots of f in I. In fact, $\text{var}(f_I)$ is also an upper bound on the number of roots of $f(x)$ in a well defined complex region. To formally state this result, we define the notion of the *Obreshkoff lens* L_m of any interval I.

The m^{th} *Obreshkoff lens* L_m of I is defined as the intersection $L_m \stackrel{\text{def}}{=\!=} \overline{C}_m \cap \underline{C}_m$ of the two open disks $\overline{C}_m, \underline{C}_m \subset \mathbb{C}$ that intersect the real axis in the endpoints a and b of I, and whose centers see the line segment (a, b) under the angle $\theta = \frac{2\pi}{m+2}$. Also, the m^{th} *Obreshkoff area* A_m is defined as the interior of $\overline{C}_m \cup \underline{C}_m$. For an illustration, see Figure 5.1. The following Theorem 5.4 gives an upper bound on the number of roots of a real polynomial in the n^{th} *Obreshkoff lens* L_n.

Theorem 5.4 ([Eig08; Obr63; Obr03]). *Let $I = (a, b)$ be an open interval. Let L_m and A_m, with $m = 0, 1, \ldots, n$, be the Obreshkoff regions in $\mathbb{C}$ as defined in Figure 5.1. Then, for all real polynomials $f(x) \in \mathbb{R}[x]$ of degree n, it holds that (all roots are counted with multiplicity):*

1. *Number of roots of f in $L_n \leq \text{var}(f_I) \leq$ Number of roots of f in A_n.*
2. *If I_1 and I_2 are two disjoint sub-intervals of I, then $\text{var}(f_{I_1}) + \text{var}(f_{I_2}) \leq \text{var}(f_I)$.*

Lemma 5.4 ([Eig08]). *For any interval $I = (0, b)$ on the positive real axis and any real polynomial $f(x) \in \mathbb{R}[x]$, $\text{var}(f_I) \leq \text{var}(f)$.*

Proof. Let $f = \sum_{i=1}^{k} f_i x^{e_i}$, here we arrange e_i's such that $e_1 < e_2 < \ldots < e_k$. Let $\beta \geq b$ be a real number, which would be chosen suitably to prove this claim. For the interval $I' = (0, \beta)$, we have:

$$f_{I'}(x) = \sum_{i=1}^{k} f_i \beta^{e_i} (x+1)^{e_k - e_i}.$$

The coefficient of x^j in $f_{I'}(x)$ is $\sum_{i=1}^{k} f_i \beta^{e_i} \binom{e_k - e_i}{j}$. It is not hard to see that there is a choice of $\beta \geq b$ such that the sign of the constant term in $f_{I'}(x)$ is same as the sign of f_k. Similarly one can choose a large enough $\beta \geq b$ such that for $1 \leq j \leq e_k - e_{k-1}$, the sign of coefficient of x^j is the same as the sign of f_{k-1}. By similar reasoning, we can also choose large enough $\beta \geq b$ such that for $e_k - e_i < j \leq e_k - e_{i-1}$, the sign of the coefficient of x^j is the same as the sign of f_{i-1}. Thus $\text{var}(f_{I'}) = \text{var}(f)$. By Using Theorem 5.4(2), we get that $\text{var}(f_I) \leq \text{var}(f_{I'}) = \text{var}(f)$. □

If we apply Lemma 5.4 for a k-nomial f then we obtain that $\text{var}(f_I) \leq \text{var}(f) \leq k - 1$ for any interval $I \subset \mathbb{R}^+$. Using Theorem 5.4(1) and Lemma 5.4, we see that the Obreshkoff lens L_n of any open interval $I = (0, b)$ contains at most $k - 1$ roots, where k is the sparsity of given polynomial. Let us record this as a fact.

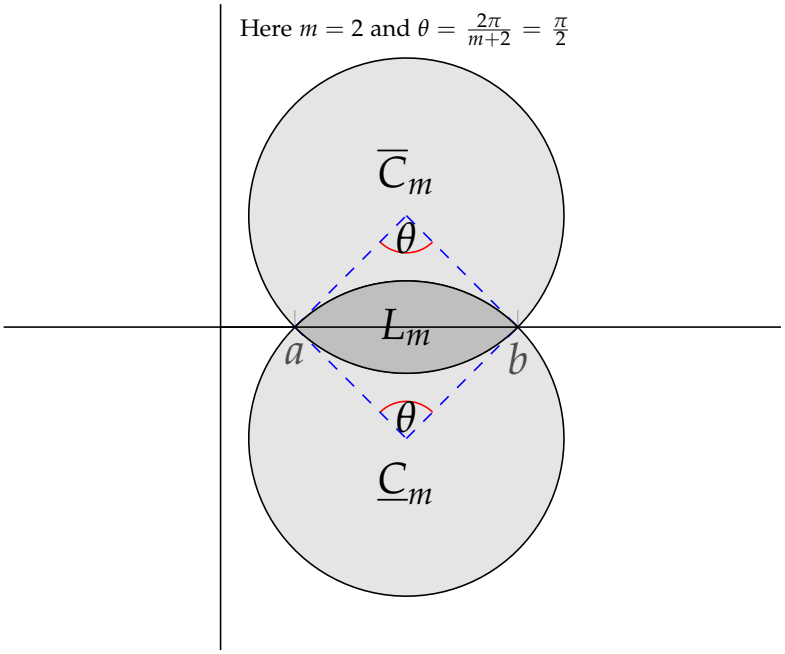

Figure 5.1: For any m, $0 \leq m \leq n$, the Obreshkoff discs $\overline{C}_m$ and $\underline{C}_m$ for $I = (a, b)$ have the endpoints of I on their boundaries; their centers see the line segment (a, b) under the angle $\theta = \frac{2\pi}{m+2}$. The Obreshkoff lens L_m is the interior of $\overline{C}_m \cap \underline{C}_m$, and the Obreshkoff area A_m is the interior of $\overline{C}_m \cup \underline{C}_m$. We have $L_n \subset \ldots \subset L_1 \subset L_0$ and $A_0 \subset A_1 \subset \ldots \subset A_n$. The cases $k = 0$ and $k = 1$ are of special interest: The circles $\overline{C}_0$ and $\underline{C}_0$ coincide. They have their centers at the midpoint of I. The circles $\overline{C}_1$ and $\underline{C}_1$ are the circumcircles of the two equilateral triangles having I as one of their edges. We call A_0 and A_1 the one and two-circle regions for I, respectively.

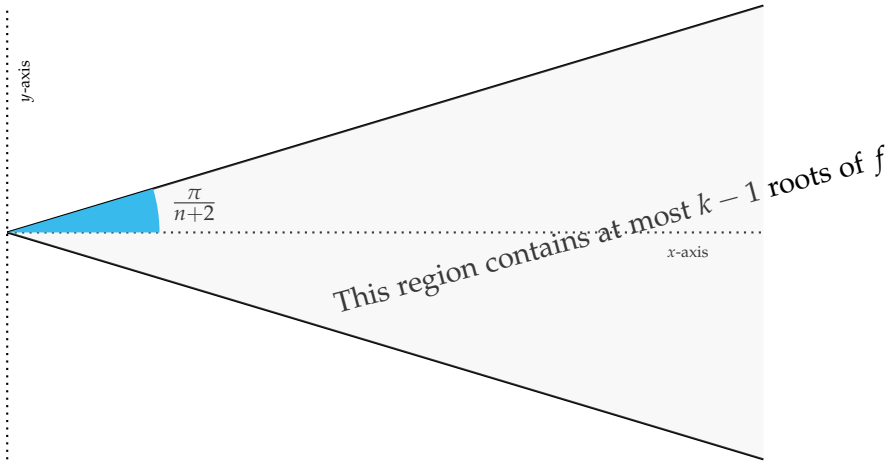

Figure 5.2: The two solid lines in the figure above meet the x-axis at the origin at an angle $\frac{\pi}{n+2}$. C_n denotes the region enclosed between these two solid lines. The cone C_n contains at most $k-1$ roots of f.

Fact 5.1. *For any k-nomial, the Obreshkoff lens L_n of any open interval on the positive real axis contains at most $k-1$ roots.*

For $b \mapsto \infty$, the Obreshkoff lens L_n of any interval $I = (0, b)$ converges to the cone C_n whose boundary are the two half-lines starting at the origin and intersecting the real axis at an angle $\pm\frac{\pi}{n+2}$; see Figure 5.2. Hence, it follows that the interior of C_n contains at most $k-1$ roots of any given real k-nomial of degree n. We prove this fact below in Theorem 5.5.

Theorem 5.5. *The cone C_n contains at most $k-1$ roots of any k-sparse real polynomial of degree n.*

Proof. Let f be a k-nomial of degree n. Assume that C_n contains more than $k-1$ roots of f. For a positive real number β, let $L_{\beta,n}$ be the Obreshkoff lens L_n of the open interval $I = (0, \beta)$. It is not hard to see that $L_{\beta,n} \subseteq C_n$ for all β. It can also be seen that $\lim_{\beta\to\infty} L_{\beta,n} = C_n$. But this contradicts the Fact 5.1. □

The real roots of a given (rational) polynomial may be arbitrary algebraic numbers and thus it might not be possible to output an explicit binary representation of all the real roots. Therefore, it is reasonable to ask only for some approximations of real roots. In particular, one wants to compute a set of disjoint "isolating" intervals for all the real roots. An interval I_α is called *isolating* for a real root α of a real polynomial $f(x) \in \mathbb{R}[x]$ if $\alpha \in I$ and I contains no other root of $f(x)$. Note that, if two real roots are very *close*, then the isolating intervals have to have end points which are also very close. This imposes a running time lower bound on any algorithm which computes such isolating intervals. We shall show that there exist integer 4-nomials which have

real roots which are very "close". In contrast, we also show that the roots of 3-nomials (also called trinomials) can not be very close.

5.4 Root Separation of Trinomials

In this section, we prove that the roots of any integer trinomial are "well-separated". We study the trinomials $f(x)$ of the form $f(x) = a_1x^{e_1} + a_2x^{e_2} + a_3$ with $e_1 > e_2$ and a_1, a_2, a_3 all being non-zero integers. Let α_1, α_2 be two distinct roots of $f(x)$. We want to show that $|\alpha_2 - \alpha_1|$ is "large". We remark that this was independently proven by Koiran [Koi17].

The overall strategy of the proof is as follows. First we shall show that any two distinct rational radicals $\sqrt[k]{\frac{a}{b}}$ and $\sqrt[\ell]{\frac{p}{q}}$ are "well-separated" (Lemma 5.6). Then, we shall complete the proof by way of contradiction. Namely, we assume that $|\alpha_2 - \alpha_1|$ is "small". By using Rolle's theorem (equivalently Lemma 5.7 for complex polynomials), we obtain a root β of $f'(x)$ which is "close" to α_1 and α_2. We see that $f'(x)$ is a binomial, thus β is either a radical or it is zero. The case of β being zero is easy to handle. So we assume that β is a radical. Then, we prove that $|f(\beta)|$ is "small" (Lemma 5.8). Thereafter, we define a new binomial $g(x)$ as: $g(x) \stackrel{\text{def}}{=\!=} f(x) - \frac{xf'(x)}{e_2} = (a_1 - \frac{a_1e_1}{e_2})x^{e_1} + a_3$. By using the fact that $|f(\beta)|$ is "small", we conclude that $|g(\beta)| = |f(\beta)|$ is also "small". By using Lemma 5.9, we obtain a root γ of $g(x)$ which is "close" to β. Now we note that β, γ are radicals multiplied with a root of unity. Thus they can not be "close" because any two distinct rational radicals are "well-separated".

In contrast to [Koi17], we also give an explicit bound on the root separation of trinomials. The first step of this proof strategy relies crucially on Theorem 5.6, which lower bounds the absolute value of a linear combination of logarithms of algebraic numbers. This is also a crucial ingredient in the proof in [Koi17]. The proof in [Koi17] proceeds by showing that a function of the form $a_1x^\beta + a_2$ can not be too small when evaluated on a rational number $\frac{p}{q}$. The analogous ingredient in our proof is that any two distinct rational radicals $\sqrt[k]{\frac{a}{b}}$ and $\sqrt[\ell]{\frac{p}{q}}$ are "well-separated". The rest of the proof in [Koi17] follows a similar structure to that of our proof. We need the following definitions.

Definition 5.4 (Mahler's measure)**.** Given a non-zero polynomial $f(x) = f_n \prod_{j=1}^{n}(x - \alpha_i) \in \mathbb{C}[x]$, its Mahler's measure (denoted by $M(f)$) is defined as:

$$M(f) \stackrel{\text{def}}{=\!=} |f_n| \prod_{j=1}^{n} \max\left(|\alpha_j|, 1\right) = |f_n| \prod_{j=1}^{n} \max_{1}\left(\alpha_j\right).$$

A related measure is the (logarithmic) height of any algebraic number α defined as below.

Definition 5.5 (Height)**.** Let α be an algebraic number of degree $n = [\mathbb{Q}(\alpha) : \mathbb{Q}]$ with $f_\alpha(x)$ being its minimal polynomial over $\mathbb{Z}$. If $f_\alpha(x) = f_n \prod_{j=1}^{n}(x - \alpha_i) \in \mathbb{Z}[x]$ with $\alpha_i \in \mathbb{C}$, then the absolute logarithmic (denoted by $h(\alpha)$) height of α is defined as:

$$h(\alpha) \stackrel{\text{def}}{=\!=} \frac{1}{n} \ln M(f_\alpha).$$

Corollary 5.4 (Height of rational numbers)**.** *For a non-zero rational number $\frac{a}{b}$, where a and b are co-prime, $h\left(\frac{a}{b}\right) = \max\{\ln|a|, \ln|b|\}$.*

Theorem 5.6 (Theorem 9.1 in [Waloo])**.** *For each $m \geq 1$, there exists a positive number $C(m)$ with the following property. Let $\lambda_1, \lambda_2, \ldots, \lambda_m$ be $\mathbb{Q}$-linearly independent natural logarithms of algebraic numbers; define $\alpha_j \stackrel{def}{=\!=} e^{\lambda_j}$ $(1 \leq j \leq m)$. Let $\beta_0, \beta_1, \beta_2, \ldots, \beta_m$ be algebraic numbers, not all of which are zero. Denote by D the degree of the number field $\mathbb{Q}(\alpha_1, \alpha_2, \ldots, \alpha_m, \beta_0, \beta_1, \beta_2, \ldots, \beta_m)$ over $\mathbb{Q}$. Further, let $B, E, E^*, A_1, A_2, \ldots, A_m$ be positive real numbers, satisfying the following conditions.*

$$\begin{aligned}
\min(B, E, E^*) &\geq e \\
\ln A_j &\geq \max\left(h(\alpha_j), \frac{E\,|\lambda_j|}{D}, \frac{\ln E}{D}\right) \\
\ln E^* &\geq \max\left(\ln D - \ln\ln E, \frac{\ln E}{D}\right) \\
B &\geq E^* \\
B &\geq \max_{1\leq i\leq m} \frac{D \ln A_i}{\ln E} \\
\ln B &\geq \max_{0\leq i\leq m} h(\beta_i)
\end{aligned}$$

Then, the number

$$\Lambda \stackrel{def}{=\!=} \beta_0 + \beta_1\lambda_1 + \ldots \beta_m\lambda_m$$

is non-zero and has absolute value bounded from below by

$$|\Lambda| \geq \exp\{-C(m)D^{m+1}(\ln B)(\ln A_1)(\ln A_2)\ldots(\ln A_m)(\ln E^*)(\ln E)^{-m-1}\}.$$

One can assign $C(m) = 2^{26m}m^{3m}$.

Now we demonstrate that Theorem 5.6 can be used to lower bound the distance between two rational radicals.

Consider $\alpha = \sqrt[k]{\frac{a}{b}}$ for positive integers a and b and similarly $\beta = \sqrt[\ell]{\frac{p}{q}}$ for positive integers p and q. We assume here that $\alpha \neq \beta$.

Lemma 5.5. *Let $\alpha = \sqrt[k]{\frac{a}{b}}$ and $\beta = \sqrt[\ell]{\frac{p}{q}}$ with $\alpha\beta \neq 1$ and a, b, p, q, k, ℓ being positive integers. If $\log(\max(a, b, p, q, k, \ell)) \leq \tau$, then $|\ln(\alpha \cdot \beta)| \geq e^{-c\cdot\tau^3}$ for some real constant $c < 2^{63}$.*

Proof. We have:

$$\ln(\alpha \cdot \beta) \quad = \quad \frac{1}{k} \ln \frac{a}{b} + \frac{1}{\ell} \ln \frac{p}{q}.$$

To lower bound this, we apply Theorem 5.6 with $m = 2$, $\beta_0 = 0, \beta_1 = \frac{1}{k}, \beta_2 = \frac{1}{\ell}, \alpha_1 = \frac{a}{b}, \alpha_2 = \frac{p}{q}$. By using Corollary 5.4, we know that $h(\alpha_1) = \max(\ln a, \ln b)$ and $h(\alpha_2) = \max(\ln p, \ln q)$. The value of D is 1. We choose $E = e$. We choose $A_1 = (\max(a, b))^e, A_2 = (\max(p, q))^e$ and $E^* = e$. If we choose $B \geq \max(k, \ell, e \ln p, e \ln q, e \ln a, e \ln b)$, then all the conditions of Theorem 5.6 are satisfied. Thus we have:

$$|\Lambda| \stackrel{\text{def}}{=\!=} |\beta_0 + \beta_1 \lambda_1 + \beta_2 \lambda_2| = |\ln(\alpha \cdot \beta)| \geq \exp\{-c' \tau^3\}. \tag{5.4}$$

Here c in the above Equation (5.4) is the constant provided by Theorem 5.6. By using Theorem 5.6 more precisely, it can be seen that any $c \leq 2^{58} \cdot e^3$ is a valid choice for c. In particular, $c < 2^{63}$. This completes the proof. □

Lemma 5.6. *Let α, β be as in Lemma 5.5 with $\alpha \neq \beta$, then $|\alpha - \beta| \geq 2^{-c' \tau^3}$ for some real constant $c' \leq 2^{64}$.*

Proof. Without loss of generality, assume that $\alpha > \beta$. We shall show $(\alpha - \beta) \geq e^{-c \cdot \tau^3} \beta$ where the constant c is the same constant as in Lemma 5.5. Let us use t to denote $e^{-c \cdot \tau^3}$. By way of contradiction, assume that $(\alpha - \beta) < t\beta$. This implies that $\left(\frac{\alpha}{\beta} - 1\right) < t$, and hence $\frac{\alpha}{\beta} < 1 + t$. This implies that $\ln \frac{\alpha}{\beta} < \ln(1 + t)$. Since $\ln(1 + x) \leq x$ for $x \geq -1$, we obtain that $\ln \frac{\alpha}{\beta} < t$. Since $\frac{1}{\beta}$ is also a radical, the inequality $\ln \frac{\alpha}{\beta} < t$ contradicts Lemma 5.5. Hence $(\alpha - \beta) \geq e^{-c \cdot \tau^3} \beta$. If $\beta > 1$ then $(\alpha - \beta) \geq e^{-c \cdot \tau^3} > 2^{-2c\tau^3}$ is trivially true. If $\beta \leq 1$ then $(\alpha - \beta) \geq 2^{-(2c \cdot \tau^3 + \log(\frac{1}{\beta}))}$. We have that $\log(\frac{1}{\beta}) = \frac{1}{\ell} \log \frac{q}{p}$. Since $p, q, \ell \geq 1$, it follows that $\log(\frac{1}{\beta}) \leq \tau$. Therefore $(\alpha - \beta) \geq 2^{-(2c \cdot \tau^3 + \tau)} \geq 2^{-(2c+1) \cdot \tau^3} \geq 2^{-2^{64} \tau^3}$. □

Remark 5.1. We have only proved that two positive radical rationals are "well-separated" but it also holds for any two rational radicals. Because if two radicals are of different sign, we can just show that they are "well-separated" from zero. More precisely, if $\alpha = -\sqrt[k]{\frac{a}{b}}$ and $\beta = \sqrt[\ell]{\frac{p}{q}}$, then we have that $|\alpha - \beta| \geq \max\{|\alpha|, |\beta|\} \geq 2^{-\tau}$.

Corollary 5.5. *Lemma 5.6 has an easy application. Suppose we want to check whether two given radicals $\alpha = \sqrt[k]{\frac{a}{b}}$ and $\beta = \sqrt[\ell]{\frac{p}{q}}$ (as in Lemma 5.5) are equal or not. Then by using Lemma 5.6, $\alpha \neq \beta$ implies that $|\alpha - \beta| \geq 2^{-2^{64} \tau^3}$. Since $\alpha = \sqrt[k]{\frac{a}{b}}$ is a solution of $bx^k = a$, we can compute $\sqrt[k]{\frac{a}{b}}$ to an error of less than 2^{-L} in time polynomial in the size of α. We can*

approximate $\beta = \sqrt[\ell]{\frac{p}{q}}$ *in a similar way. Thus we can compute sufficiently good approximations of* α *and* β *until we know for sure that either* $|\alpha - \beta| < 2^{-2^{64}\tau^3}$ *or* $|\alpha - \beta| > 0$.

5.4.1 Complex root separation

Now, we demonstrate the second step of the proof strategy described above. Recall the setup: we have an integer trinomial $f(x) = a_1 x^{e_1} + a_2 x^{e_2} + a_3$. Let α_1 and α_2 be two distinct roots of $f(x)$ with minimum separation. First we recall a complex version of Rolle's theorem. To this end, we need the following theorem from [Mar85].

Theorem 5.7 (Theorem 5.1 in [Mar85])**.** *If* z_1 *and* z_2 *are any two distinct zeros of a polynomial* f *of degree* n*, then at least one critical point (root of* f'*) of* f *lies on the circular disk* $|z - c| < r$*, where* $c = \frac{z_1+z_2}{2}$ *and* $r = \frac{|z_1-z_2|}{2} \cdot \cot(\frac{\pi}{n})$.

Lemma 5.7. *If* z_1 *and* z_2 *are any two distinct zeros of a polynomial* f *of degree* n *with* $|z_1 - z_2| \leq \epsilon$*, then there exists a root* z' *of* f' *such that* $|z' - z_i| \leq \frac{\epsilon}{2}(1 + \frac{n}{\pi})$ *for* $i \in \{1, 2\}$.

Proof. Let $|z_1 - z_2| = R$. By Theorem 5.7, we know that there exists a root of z' in the complex disc centered at $\frac{z_1+z_2}{2}$ and of radius $r = \frac{R}{2} \cdot \cot(\frac{\pi}{n})$. Thus $|z' - z_i| \leq \frac{R}{2} + \frac{R}{2} \cdot \cot(\frac{\pi}{n})$ for $i \in \{1, 2\}$. We know that $\cot(x) \leq \frac{1}{x}$ for $0 \leq x \leq \pi$. Thus $|z' - z_i| \leq \frac{R}{2}(1 + \frac{n}{\pi}) \leq \frac{\epsilon}{2}(1 + \frac{n}{\pi})$. □

The following Lemma 5.8 demonstrates that if we evaluate an (n, k, τ)-nomial at some point z which is "close" to a root z_0 of polynomial f then $|f(z)|$ is "small".

Lemma 5.8. *Let* $f \in \mathbb{R}[x]$ *be an* (n, k, τ)*-nomial and let* $z \in \mathbb{C}$ *be such that there exists a root* z_0 *of* f *with* $|z - z_0| \leq \epsilon$*, then* $|f(z)| \leq \epsilon \cdot 2^{n \cdot \log \max(1,|z|,|z_0|)+\tau+\log k+\log n}$.

Proof. Define $M := \max(1, |z|, |z_0|)$. For every positive integer m, we have:

$$z^m - z_0^m = (z - z_0)(\sum_{j=0}^{m-1} z^j z_0^{m-1-j}).$$

Thus $|z^m - z_0^m| \leq \epsilon \cdot m \cdot M^{m-1}$. If $f(x) = \sum_{i=1}^{k} a_i x^{e_i}$ then,

$$f(z) - f(z_0) = f(z) = \sum_{i=1}^{k} a_i (z^{e_i} - z_0^{e_i}).$$

By using $a_i \leq 2^\tau$ and $|z^m - z_0^m| \leq \epsilon \cdot m \cdot M^{m-1}$, we get that:

$$|f(z)| \leq \epsilon \cdot k \cdot n \cdot 2^\tau \cdot M^{n-1}.$$

Thus the claimed bound follows. □

To complete our proof of separation of complex roots of Trinomials, we also need the converse of Lemma 5.8 for binomials.

Lemma 5.9. *Let $f(x) = a + bx^n$ be an integer binomial with complex roots $z_1, z_2, \ldots, z_n$ and $r = \left|\frac{a}{b}\right|^{\frac{1}{n}}$. Let $t \geq 1$ be any real number. If $z \in \mathbb{C}$ satisfies $\min_k |z - z_k| > \frac{2r}{nt}$, then $|f(z)| \geq \frac{1}{nt}$.*

Proof. All the roots $z_1, z_2, \ldots, z_n$ of $f(x)$ lie on the circle in the complex plane with radius $r = \left|\frac{a}{b}\right|^{\frac{1}{n}}$, separated equally by angles of $\frac{2\pi}{n}$. Now define the n disks $\Delta_1, \Delta_2, \ldots, \Delta_n$: $\Delta_i \stackrel{\text{def}}{=\!=} \Delta_{\frac{2r\sin(\frac{\pi}{n})}{nt}}(z_i)$. Therefore for any root z_i of $f(x)$, we have $\sigma(z_i, f) = 2r\sin(\frac{\pi}{n})$.

We note that f is holomorphic on $\mathbb{C} \setminus \cup_{i\in[n]}\Delta_i$. Hence $|f(z)|$ can attain minimum only at the boundary of $\mathbb{C} \setminus \cup_{i\in[n]}\Delta_i$. Therefore the minimum of $|f(z)|$ for $z \in \mathbb{C} \setminus \cup_{i\in[n]}\Delta_i$ is achieved when z is on the boundary of some Δ_i. Since $z \in \mathbb{C} \setminus \cup_{i\in[n]}\Delta_i$, we obtain that $|f(z)| \geq |f(y)|$ for all y lying on the boundaries of Δ_i's. Hence to prove the claim, we only need to prove that $|f(y)| \geq \frac{1}{nt}$ whenever y is on the boundary of some disc Δ_k. We have that $f(x) = b\prod_{1\leq i\leq n}(x - z_i)$. Suppose y is on the boundary of some disc Δ_k. Therefore:

$$
\begin{aligned}
|f(y)| &= |b| \cdot \prod_{1\leq i\leq n} |y - z_i| = |b| \cdot |y - z_k| \prod_{1\leq i\leq n, i\neq k} |y - z_i| \\
&= |b| \cdot \left(\prod_{1\leq i\leq n, i\neq k} |z_k - z_i|\right) \cdot |y - z_k| \cdot \prod_{1\leq i\leq n, i\neq k} \frac{|y - z_i|}{|z_k - z_i|} \\
&= |f'(z_k)| \cdot |y - z_k| \cdot \prod_{1\leq i\leq n, i\neq k} \frac{|y - z_i|}{|z_k - z_i|} \\
&\geq |b| \cdot n \cdot |z_k|^{n-1} \cdot \frac{2r\sin(\frac{\pi}{n})}{nt} \cdot \prod_{1\leq i\leq n, i\neq k} \frac{|z_k - z_i| - |y - z_k|}{|z_k - z_i|} \\
&\geq |a| \cdot \frac{2r\sin(\frac{\pi}{n})}{t \cdot |z_k|} \cdot \left(1 - \frac{1}{tn}\right)^{n-1} \geq \frac{2\sin(\frac{\pi}{n})}{et} \\
&\geq \frac{1}{2} \cdot \frac{2}{\pi} \cdot \frac{\pi}{n} \cdot \frac{1}{t} \geq \frac{1}{nt}. \qquad (\tfrac{2}{\pi}x \leq \sin(x) \leq x \quad \text{for } 0 \leq x \leq \tfrac{\pi}{2})
\end{aligned}
$$

□

Now we are ready to prove that even the complex roots of an integer trinomial are well separated.

Theorem 5.8. Let $f(x) = a_1x^{e_1} + a_2x^{e_2} + a_3$ *be an integer trinomial satisfying the condition:* $\log\max(e_1, e_2, |a_1|, |a_2|, |a_3|) \leq \tau$ *and* $e_2 \leq e_1 = n$. *If z_1 and z_2 are two distinct roots of $f(x)$ then $|z_1 - z_2| \geq 2^{-c\tau^3}$* for some $c < 2^{68}$.

Proof. We first assume that $|z_1| \leq 1$ or $|z_2| \leq 1$, with $|z_1 - z_2| = \delta$. Without loss of generality, assume that $|z_1| \leq 1$. We want to prove that $\delta \geq 2^{-c\tau^3}$. By way of contradiction, assume that $\delta < 2^{-c\tau^3}$.

By using Lemma 5.7, we know that there exists a root z' of $f'(x)$ such that:

$$|z' - z_i| \leq \frac{\delta}{2}(1 + \frac{n}{\pi}) \text{ for } i \in \{1, 2\}.$$

Thus $|z'| \leq |z_1| + \frac{\delta}{2}(1 + \frac{n}{\pi})$. Since $\delta < 2^{-c\tau^3}$, we get that $\delta \leq \frac{1}{2n^3}$, thus $|z'| \leq 1 + \frac{1}{n}$. Therefore

$$n \cdot \log \max(1, |z'|, |z_1|) \leq 2.$$

By using Lemma 5.8 on z_1 and z', we know that:

$$\begin{array}{lll} |f(z')| & \leq & \frac{\delta}{2}(1 + \frac{n}{\pi}) \cdot 2^{n \cdot \log \max(1, |z'|, |z_1|) + \tau + \log 3 + \log n}. \\ & \leq & \frac{\delta}{2}(1 + \frac{n}{\pi}) \cdot n \cdot 2^{2 + \tau + \log 3} < \frac{\delta}{2}(1 + \frac{n}{\pi}) \cdot n \cdot 2^{4+\tau}. \\ < \delta \cdot n^2 . 2^{4+\tau} & < 2^{-c\tau^3 + 4 + \tau + 2\log n} < 2^{-(c+7)\tau^3}. & \end{array}$$

Now define $g(x) \stackrel{\text{def}}{=\!=} f(x) - \frac{xf'(x)}{e_2} = (a_1 - \frac{a_1 e_1}{e_2})x^{e_1} + a_3$. Since, $f'(z') = 0$, we get that $|g(z')| = |f(z')| < 2^{-(c+7)\tau^3}$. We choose a positive real number t such that $|f(z')| < \frac{1}{nt}$, *i.e.*, we want that the condition $|f(z')| \geq \frac{1}{nt}$ of Lemma 5.9 is false. Since $|f(z')| < 2^{-(c+7)\tau^3}$, we choose $t = \frac{2^{(c+7)\tau^3}}{2n}$.

We now apply Lemma 5.9 on $g(x)$, t and z'. This gives us a root z'' of $g(x)$ such that $|z' - z''| < \frac{2r}{e_1 t}$, with $r = \left|\frac{a_3 e_2}{a_1 e_1 - a_1 e_2}\right|^{\frac{1}{e_1}}$. In particular

$$|z' - z''| < \frac{4rn}{e_1} \cdot 2^{-(c+7)\tau^3} < 2^{-(c+7)\tau^3 + 2 + \tau + 2\tau} < 2^{-2^{67}\tau^3}.$$

Now note that $z'' = \zeta r$, where ζ is a root of unity. Also, $z' = 0$ or $z' = \xi r'$ with $r' = \left|\frac{a_2 e_2}{a_1 e_1}\right|^{\frac{1}{e_1 - e_2}}$ and ξ being a root of unity. Lemma 5.6 shows that difference of two real rational radicals is at least $2^{-c'(2\tau)^3}$ for some $c' < 2^{64}$. But this also implies that $|z' - z''| \geq 2^{-c''\tau^3}$ for some $c'' < 2^{67}$. Thus our assumption was wrong. Hence $\delta \geq 2^{-c\tau^3}$.

Here we assumed that $|z_1| \leq 1$ or $|z_2| \leq 1$. Suppose we have $|z_1| \geq 1$ and $|z_2| \geq 1$. Then we look at the polynomial:

$$h(x) = x^{e_1} f(\frac{1}{x}) = a_1 + a_2 x^{e_1 - e_2} + a_3 x^{e_1}.$$

Corresponding to z_1 and z_2, $\frac{1}{z_1}$ and $\frac{1}{z_2}$ are roots of $h(x)$. Also, $\left|\frac{1}{z_1}\right| \leq 1$ or $\left|\frac{1}{z_2}\right| \leq 1$. Thus applying first part of proof of $h(x)$, we get that

$$\left|\frac{1}{z_1} - \frac{1}{z_2}\right| = \left|\frac{z_2 - z_2}{z_1 z_2}\right| \geq 2^{-c\tau^3}.$$

Since $|z_1| \geq 1$ and $|z_2| \geq 1$, we get that $|z_1 - z_2| \geq 2^{-c\tau^3}$. □

5.5 Root separation for 4-nomials

We have shown that the roots of integer trinomials are well-separated. This might lead one to hope that the roots of other integer fewnomials are also well-separated. We demonstrate an example of 4-nomials which shows that this is not the case in general. Moreover we also prove that to isolate real roots of these 4-nomials, we need exponentially many bits. We need the Rouché's Theorem (Theorem 5.9).

Theorem 5.9 (Rouché's Theorem, Theorem 3.8 in [Con78])**.** *Let f and g be holomorphic inside some region Δ with boundary $\partial\Delta$. If $|f(z)| > |f(z) - g(z)|$ on $\partial\Delta$, then f and g have the same number of zeros inside Δ.*

The following Lemma 5.10 and Lemma 5.12 were already proved in [Sag14] but we simplify the proofs of [Sag14] here. Lemma 5.12 was proved using continued fractions in [Sag14]. We give here a much simpler proof of Lemma 5.12 .

Lemma 5.10. *If $a > 16$ and $n > 4$ are positive integers, then there exist exactly two roots of the 4-nomial $x^n - (ax^2 - 1)^2$ in $\Delta_r\left(\frac{1}{\sqrt{a}}\right)$ for $r = \left(\frac{2}{\sqrt{a}}\right)^{\frac{n}{2}}$.*

Proof. We use Theorem 5.9 on $f = -(ax^2 - 1)^2$ and $g = x^n - (ax^2 - 1)^2$ with $\Delta \stackrel{\text{def}}{=\!=} \Delta_r\left(\frac{1}{\sqrt{a}}\right)$. We have that $f - g = -x^n$. Thus we need to prove that $\left|(az^2 - 1)^2\right| > |z^n|$ for all $z \in \partial\Delta$. First, we show that $r < \frac{1}{\sqrt{a}}$. We have that $r = \frac{1}{\sqrt{a}} \cdot 2^{\frac{n}{2}} \cdot \left(\frac{1}{\sqrt{a}}\right)^{\frac{n}{2}-1}$. The condition $a > 16, n > 4$ implies that $2^{\frac{n}{2}} \cdot \left(\frac{1}{\sqrt{a}}\right)^{\frac{n}{2}-1} < 1$. Thus $r < \frac{1}{\sqrt{a}}$.

The maximum value of $|z^n|$ for $z \in \partial\Delta$ is achieved when $z = r + \frac{1}{\sqrt{a}}$. Since $r < \frac{1}{\sqrt{a}}$, we get that $|z^n| = \left(r + \frac{1}{\sqrt{a}}\right)^n < \left(\frac{2}{\sqrt{a}}\right)^n$. Similarly, the minimum value of $\left|(az^2 - 1)^2\right|$ on Δ is achieved when $z = r - \frac{1}{\sqrt{a}}$. For $z = r - \frac{1}{\sqrt{a}}$, we have that $\left|(az^2 - 1)^2\right| = ar^2(-2 + \sqrt{a}r)^2 \geq ar^2 \geq a\left(\frac{2}{\sqrt{a}}\right)^n$. Thus $\left|(az^2 - 1)^2\right| > |z^n|$ for all $z \in \partial\Delta$. Since f has the zero $\frac{1}{\sqrt{a}}$ inside Δ with multiplicity two, by using Theorem 5.9 we get that $x^n - (ax^2 - 1)^2$ also has exactly two roots inside Δ. □

Lemma 5.10 shows that there exist exactly two roots of $x^n - (ax^2 - 1)^2$ in $\Delta_r(\frac{1}{\sqrt{a}})$. But it does not say anything about the nature of these roots. We prove that these roots are real.

Lemma 5.11. *If $a > 16$ and $n > 4$ are positive integers then there exist two distinct real roots of the 4-nomial $x^n - (ax^2 - 1)^2$ in $\Delta_r\left(\frac{1}{\sqrt{a}}\right)$ for $r = \left(\frac{2}{\sqrt{a}}\right)^{\frac{n}{2}}$.*

Proof. Let $g \stackrel{\text{def}}{=\!=} x^n - (ax^2-1)^2$. It is easy to check that $g\left(\frac{1}{\sqrt{a}}\right) = \left(\frac{1}{\sqrt{a}}\right)^n > 0$. First we check the sign of g at $\frac{1}{\sqrt{a}} - r$. We have:

$$\begin{aligned} g\left(\frac{1}{\sqrt{a}} - r\right) &= \left(\frac{1}{\sqrt{a}} - r\right)^n - \left(a\left(\frac{1}{\sqrt{a}} - r\right)^2 - 1\right)^2 \\ &= \left(\frac{1}{\sqrt{a}} - r\right)^n - ar^2\left(-2 + \sqrt{a}r\right)^2 \\ &< \left(\frac{1}{\sqrt{a}}\right)^n - ar^2 < 0. \qquad \text{(By using } r < \tfrac{1}{\sqrt{a}} \text{ and } r = \left(\tfrac{2}{\sqrt{a}}\right)^{\frac{n}{2}}\text{)} \end{aligned}$$

Thus g has a real root $\alpha \in \left(\frac{1}{\sqrt{a}} - r, \frac{1}{\sqrt{a}}\right)$.

Now we check the sign of g at $\frac{1}{\sqrt{a}} + r$. We have the following equality:

$$\begin{aligned} g\left(\frac{1}{\sqrt{a}} + r\right) &= \left(\frac{1}{\sqrt{a}} + r\right)^n - \left(a\left(\frac{1}{\sqrt{a}} + r\right)^2 - 1\right)^2 \\ &= \left(\frac{1}{\sqrt{a}} + r\right)^n - ar^2\left(2 + \sqrt{a}r\right)^2 \\ &< \left(\frac{2}{\sqrt{a}}\right)^n - 4a\left(\frac{2}{\sqrt{a}}\right)^n < 0. \qquad \text{(By using } r < \tfrac{1}{\sqrt{a}} \text{ and } r = \left(\tfrac{2}{\sqrt{a}}\right)^{\frac{n}{2}}\text{)} \end{aligned}$$

Thus g has a real root $\beta \in \left(\frac{1}{\sqrt{a}}, \frac{1}{\sqrt{a}} + r\right)$. Hence there exists two real roots of g in $\left(\frac{1}{\sqrt{a}} - r, \frac{1}{\sqrt{a}} + r\right)$. □

Now we prove that $\frac{1}{\sqrt{a}}$ can not be approximated by a rational number of small bit length.

Lemma 5.12. *If a is a positive integer such that $\sqrt{a} \notin \mathbb{Z}$, then $\left|\frac{1}{\sqrt{a}} - \frac{p}{q}\right| > \frac{1}{4apq^2}$ for any $p, q \in \mathbb{Z}$ and $q \neq 0$.*

Proof. Let $\epsilon = \frac{1}{\sqrt{a}} - \frac{p}{q}$ and suppose that $\epsilon \geq 0$. We can assume that $\epsilon > 0$ because otherwise $\frac{1}{\sqrt{a}} - \frac{p}{q} = 0$. This implies that $\sqrt{a} \in \mathbb{Z}$, a contradiction. We also observe that we can assume $\frac{p}{q} > 0$. Otherwise, $\left|\frac{1}{\sqrt{a}} - \frac{p}{q}\right| > \frac{1}{\sqrt{a}} > \frac{1}{4apq^2}$. Therefore it can be assumed that $p, q > 0$. We have the following equality:

$$\begin{aligned} \frac{1}{a} - \frac{p^2}{q^2} &= \left(\frac{1}{\sqrt{a}} - \frac{p}{q}\right) \cdot \left(\frac{1}{\sqrt{a}} + \frac{p}{q}\right) \\ &= \epsilon \cdot \left(\epsilon + \frac{2p}{q}\right) = \epsilon^2 + \frac{2p\epsilon}{q} \end{aligned}$$

This implies:

$$q^2 - ap^2 = aq^2\epsilon^2 + 2apq\epsilon.$$

Note that $q^2 - ap^2$ is an integer. If $\epsilon < \frac{1}{4apq}$, then $\left|aq^2\epsilon^2 + 2apq\epsilon\right| < 1$. Thus $\frac{1}{\sqrt{a}} - \frac{p}{q} \geq \frac{1}{4apq}$.

Now we look at the case when $\frac{p}{q} - \frac{1}{\sqrt{a}} > 0$. Let $\delta = \frac{p}{q} - \frac{1}{\sqrt{a}}$. By applying the similar calculations as above we obtain the following equality:

$$ap^2 - q^2 = aq^2\delta^2 + 2\sqrt{a}\delta q^2.$$

If $\delta < \frac{1}{4\sqrt{a}q^2}$ then $aq^2\delta^2 + 2\sqrt{a}\epsilon\delta q < 1$. Thus $\frac{p}{q} - \frac{1}{\sqrt{a}} \geq \frac{1}{4\sqrt{a}q^2}$. Hence $\left|\frac{1}{\sqrt{a}} - \frac{p}{q}\right| > \min\left(\frac{1}{4apq}, \frac{1}{4\sqrt{a}q^2}\right) \geq \frac{1}{4apq^2}$. □

Theorem 5.10. *Any algorithm which isolates the real roots of $f(x) = x^n - (2^{2\tau}x^2 - 1)^2$ requires $\Omega(n\tau)$ bit operations, here $\tau > 4$.*

Proof. By using Lemma 5.10 on $a = 2^{2\tau}$, we get that f has two real roots inside $\Delta = \Delta_r(2^{-\tau})$ for $r = (2^{1-\tau})^{\frac{n}{2}}$. Any algorithm which outputs an isolating interval which separates roots inside Δ will have as an end point a rational number $\frac{p}{q}$ such that $\left|\frac{1}{\sqrt{a}} - \frac{p}{q}\right| < r$. By using Lemma 5.12, we know that $\frac{1}{4apq^2} < r$. Thus $pq^2 > \frac{1}{4ar} = 2^{\frac{n}{2}(\tau-1)-2\tau-2}$. Thus $\max(p,q) = 2^{\Omega(n\tau)}$. Hence even to encode p, q, any such isolating algorithm would need $\Omega(n\tau)$ many bit operations. □

5.6 Introduction and History of Root Computation

We saw above that even in the case of 4-nomials, there is no hope for a polynomial time (in the "input size" $O(k(\log n + \tau))$) algorithm for isolating its real roots. Thus, we shall define a more relaxed notion of the computation of roots for k-nomials and demonstrate a polynomial time algorithm for computing such a relaxation.

How does one specify the input polynomial to root computation algorithms? If the input polynomial is an integer polynomial then the list of coefficients can be provided as an input. Otherwise, we assume that the coefficients are given by aid of an oracle. More specifically, we assume the existence of an oracle that provides arbitrary good approximations of the given (n, k, τ)-nomial f: Let $L \geq 1$ be an integer. We call a polynomial $\tilde{f} = \tilde{f}_n x^n + \ldots + \tilde{f}_0$, with $\tilde{f}_i = s_i \cdot 2^{-(L+1)}$ and $s_i \in \mathbb{Z}$, an *absolute L-approximation* of f if $\left|\tilde{f}_i - f_i\right| \leq 2^{-L}$ for all i. We assume that we can obtain such an approximation $\tilde{f}$ at $O(k(\log n + L + \tau))$ cost. This is the cost of reading the coefficients of $\tilde{f}$. We frequently use the phrase "the coefficients of f need to be approximated to L bits after the binary points" instead of "the algorithm requires an absolute L-approximation of f".

For a survey of literature on root computation algorithms, see [McN07; MP13; Pan97; SM16; Bec+18]. In whatever follows, this chapter deals with computing the real roots of k-nomials. All the known root computation algorithms can be divided into the two following categories.

1. Iterative algorithms for simultaneously computing (or some approximation of) all the roots.

2. Subdivision methods in which one starts with a region containing all the roots and then subdivide this region according and check whether a region contains exactly one root or no root.

We solely focus on subdivision algorithms in this chapter. The famous examples of the subdivision methods are the Descartes's method and the splitting circle method. See [CL76; Meh+06; Eig08; RZ04] for some examples of the Descartes's method.

The splitting circle method was introduced by Schönhage in [Sch82]. It was further improved by Pan in [Pan02]. The algorithm in [Pan02] isolates all the complex roots of a polynomial. For integer polynomials $f(x)$ of degree n and coefficients magnitude bounded by 2^τ, Pan's algorithm isolates all the complex roots using $\tilde{O}\left(n^2\tau\right)$ bit operations. This algorithm of [Pan02] essentially runs the factorization algorithm where the precision is controlled by the worst case root separation bound $\sigma(f)$, which is $2^{-\Theta(n(\log n+\tau))}$ [Mah64]. Pan's algorithm is not adaptive with respect to the root separation of the input polynomial and assumes the worst case root separation bound of $2^{-\Theta(n(\log n+\tau))}$. An algorithm that determines the precision adaptively was described in [MSW15]. In the worst case root separation, the algorithm in [MSW15] runs in time $\tilde{O}\left(n^3+n^2\tau\right)$. For integer polynomials, this running time essentially matches that of Pan.

All the algorithms mentioned above, compute the complex roots of polynomials. In contrast, there exist dedicated algorithms which only compute real roots. These algorithms have an advantage of usually being much simpler. the Descartes method is one such method which can (only) be used to isolate the real roots. See [ESY06] for a standard example of the Descartes method to isolate the real roots. For integer polynomials $f(x)$ of degree n and coefficients magnitude bounded by 2^τ, the algorithm of [ESY06] isolates all the real roots in time $\tilde{O}\left(n^4\tau^2\right)$. One can even use approximate arithmetic in Descartes method. This allows one to compute real root approximations of real polynomials also, whose coefficients are given by an oracle.

Approximate arithmetic with Descartes's method was used in [Meh+06] to describe an algorithm which isolates all the real roots of a square-free real polynomial $f(x) = \sum_{i=0}^{n} f_i x^i$, coefficients $2^{-\tau} \leq |f_i| \leq 2^\tau$, with $O(n^4(\tau - \log(\sigma(f)))^2)$ bit operations, this running time was improved to $\tilde{O}\left(n^3+n^2\tau\right)$ in [SM16].

Another common problem in root computation is the refinement. Here one is given a real interval I which already isolates a real root α of the given polynomial $f(x)$. But we want to find a sub-interval I' of I such that $\alpha \in I'$ with $|I'| \leq 2^{-L}$, where L is

an additional integer input to the refinement routine. Sagraloff [Sag14] combines the Newton iteration and bisection to refine the intervals. However, the refinement method of [Sag14] only applies to rational polynomials. [SM16] uses the idea of [Sag14] along with approximate computation and the choice of admissible points, to demonstrate a refinement routine for arbitrary coefficients.

All the methods (except [Sag14]) above are not sparsity aware. That is, if the input polynomial is an (n, k, τ)-nomial then the above algorithms do not take advantage of the input polynomial being sparse. Sagraloff [Sag14] uses refinement to demonstrate an algorithm which for integer (n, k, τ)-nomials, isolates all the real roots in $\mathsf{poly}(O(k \cdot (\log n + \tau)))$ arithmetic operations. But the bit complexity of the algorithm in [Sag14] is still $\tilde{O}\left(k^4 \cdot n\tau\right)$, which is essentially optimal for small k, as shown in Theorem 5.10.

For (n, k, τ)-nomials, all the algorithms mentioned in above discussion do not run in polynomial time, *i.e.*, in time $\mathsf{poly}(O(k \cdot (\log n + \tau)))$. Suppose we only want to compute integer or rational roots of a given integer (n, k, τ)-nomial. In this case, polynomial time algorithms are known (see [CKS99; LJ99]). We describe the algorithm of [CKS99] in Section 5.7.

5.7 Fractional Derivatives and Integer Roots

In this section, we introduce yet another method to compute the real roots. In this method, one first computes the real roots (or some approximations) of $f'(x)$. By using Rolle's theorem, we know that there is at least one real root $\alpha \in (a, b)$ of $f'(x)$ if a, b are the real roots of $f(x)$. Thus if α, β are some roots of $f'(x)$ with no root of $f'(x)$ being in (α, β), then (α, β) can contain at most one root of $f(x)$. And (α, β) contains a root of $f(x)$ if and only if $f(\alpha)f(\beta) < 0$. Thus, one can compute the real roots of $f(x)$ from the real roots of $f'(x)$. Similarly, we can use recursion to compute the real roots of $f'(x)$. We have omitted several issues here, like the following.

1. It is usually not possible to compute the roots of $f'(x)$ exactly.
2. Checking $f(\alpha)f(\beta) < 0$ might be too costly. This happens when $|f(\alpha)|$ or $|f(\beta)|$ might be very small and thus we might need a very high precision to compute the signs of $f(\alpha)$ and $f(\beta)$.

Now we demonstrate an application of the above strategy, which was first described in [CKS99]. The following Problem 5.1 was solved in [CKS99].

Problem 5.1 (Integer roots of Integer (n, k, τ)-nomials)**.** *Given an integer (n, k, τ)-nomial $f(x) \in \mathbb{Z}[x]$, compute all integer roots of $f(x)$.*

Note that the size of the sparse representation of any integer (n, k, τ)-nomial is $O(k \cdot (\tau + \log n))$.

Theorem 5.11 (Theorem 1 in [CKS99]). *There is a polynomial time algorithm for Problem 5.1, i.e., it runs in* $\mathsf{poly}(k \cdot (\tau + \log n))$ *bit operations.*

To prove Theorem 5.11, we introduce the following definition:

Definition 5.6 (Fractional derivative). Let f be a k-nomial of the form:

$$f(x) = \sum_{i=1}^{k} f_i x^{e_i} \in \mathbb{R}[x].$$

Here e_i's are non-negative integers, with $0 = e_1 < e_2 < \ldots < e_k \leq n$. Then, we define $f^{[1]} \stackrel{\text{def}}{=\!=} \frac{f'}{x^{e_2-1}}$ as the *(first) fractional derivative of* f. In other words, we divide the first derivative f' of f by the highest possible power of x that divides f'.

The i^{th} *fractional derivative* $f^{[i]}$ *of* f is then recursively defined as the first fractional derivative of $f^{[i-1]}$. Notice that if $f(x)$ is an (n, k, τ)-nomial then, for $i \leq k-1$, $f^{[i]}$ is an $(n, k-i, \tau + i \cdot \log n)$-nomial with a non-zero constant term and $f^{[i]} = 0$ for $i \geq k$. We further use the notation $\mathcal{D}_f$ to denote the tuple of all non-zero fractional derivatives $f, f^{[1]}, f^{[2]}, \ldots, f^{[k-1]}$, *i.e.*, $\mathcal{D}_f = (f, f^{[1]}, f^{[2]}, , \ldots, f^{[k-1]})$.

Now we restate the definition of locating intervals as given in [CKS99].

Definition 5.7. A list $L = \{[u_i, v_i]\}_{i \in [N]}$ of closed intervals with integer end points and satisfying $u_i \leq v_i \leq u_{i+1}$, is said to locate the roots of $f(x) \in \mathbb{Z}[x]$ in the interval $[-M, M]$ if for each root $\alpha \in [-M, M]$ of $f(x)$, there exists an $i \in [N]$ such that $\alpha \in [u_i, v_i]$.

We also call such lists to be locating lists for the roots of f. We further state the following Theorem 5.12 from [CKS99].

Theorem 5.12 (Theorem 1 in [CKS99]). *There exists an algorithm which, for a given* $a \in \mathbb{Z}$ *and an integer* (n, k, τ)*-nomial* $f(x) \in \mathbb{Z}[x]$*, computes the sign of* $f(a)$ *in* $\mathsf{poly}(k \cdot (\tau + \log n), \log(|a|) + 1)$ *bit operations.*

For computing the integer roots, we further need a "refinement" routine. Here, we are given an isolating integral interval $[a, b]$, *i.e.*, $a, b \in \mathbb{Z}$ and $[a, b]$ contains exactly one real root (this root might not be an integer) α. The desired "refinement" routine would refine $[a, b]$ to an isolating integral interval I of length at most 1 such that $\alpha \in I$. For the rest of this section, we define $B \stackrel{\text{def}}{=\!=} 2^{2\tau} + 1$.

Lemma 5.13. *Algorithm 5.5 refines the isolating interval* $[a, b]$ *(with* $\log(\max(a, b)) = O(\tau)$*) in* $\mathsf{poly}(\tau, \log n, k)$ *bit operations.*

Proof. Note that Algorithm 5.5 in line 17 and line 19 can recurse at most $O(\log(B)) = O(\tau)$ many times. Also, the sign computations in line 6 can be performed using $\mathsf{poly}(\tau, \log n, k)$ bit operations, this follows from Theorem 5.12. Thus Algorithm 5.5 uses $\mathsf{poly}(\tau, \log n, k)$ bit operations. The correctness also follows from the fact if we have not terminated by the line 16, then only one of the subdivided intervals in line 17 and line 19 can contain the root α. □

Algorithm 5.5 Refine an isolating interval.

Input: An integer (n, k, τ)-nomial $f(x) \in \mathbb{Z}[x]$ and an isolating interval $[a, b]$ (with $a, b \in \mathbb{Z}$ and $|a|, |b| \leq B$) for some real root α of f.
Output: An integral sub-interval I of $[a, b]$ of length at most 1 such that $\alpha \in I$.

if $b \leq a + 1$ **then**
    return $[a, b]$
end if
$c_1 \leftarrow \lfloor \frac{a+b}{2} \rfloor$
$c_2 \leftarrow \lceil \frac{a+b}{2} \rceil$
Compute the sign of $f(x)$ at a, c_1, c_2, b.
if $f(a) = 0$ **then**
    return $[a, a]$
else if $f(b) = 0$ **then**
    return $[b, b]$
else if $f(c_1) = 0$ **then**
    return $[c_1, c_1]$
else if $f(c_2) = 0$ **then**
    return $[c_2, c_2]$
end if
if $f(a)f(c_1) < 0$ **then**
    return output of this Algorithm 5.5 on f and $[a, c_1]$
else
    return output of this Algorithm 5.5 on f and $[c_2, b]$
end if

The following Algorithm 5.6 computes a locating list for f if one has already computed a locating list for $f^{[1]}$.

Algorithm 5.6 Compute locating list of $f(x)$.

Input: An integer (n, k, τ)-nomial $f(x) \in \mathbb{Z}[x]$ and a locating list L' for $f^{[1]}$ in $(-B, B)$. It is assumed that all intervals in L' have width 0 or 1.
Output: A locating list for $f(x)$ in $(-B, B)$.

1: $L' \leftarrow L' \cup \{[0, 0]\}$
2: Assume $L' = \{[u_i, v_i]\}_{i \in [N]}$
3: $L \leftarrow L'$
4: Using the algorithm of Theorem 5.12, compute the signs of $f(x)$ at points $-B, u_1, v_1, \ldots, u_N, v_N, B$.
5: **for** each interval $[a, b] \in \{[-B, u_1], [v_1, u_2], \ldots, [v_N, B]\}$ **do**
6: **if** $f(a)f(b) < 0$ **then**
7: $I \leftarrow$ output of Algorithm 5.5 on f and $[a, b]$
8: $L \leftarrow L \cup I$
9: **end if**
10: **end for**
11: **return** L

Proposition 5.1 (Proposition 1 in [CKS99]). *For any integer (n, k, τ)-nomial $f(x) \in \mathbb{Z}[x]$, Algorithm 5.6 computes a locating list in $(-B, B)$ for f of size at most $N + 2k$ and works in $\mathsf{poly}(\tau, \log n, k, N)$ bit operations.*

Proof. Note that all the roots of $f^{[1]}$ and f' are the same except that f' might have zero as an additional root. Thus the locating list L' in line 1 is a locating list for f'. Computations in line 4 of Algorithm 5.6 have $\mathsf{poly}(\tau, \log n, k, N)$ cost, this follows from Theorem 5.12. Let $[a, b]$ be some interval in line 5. The corresponding open interval (a, b) can contain at most one root of f, because otherwise L' (in line 1) could not be a locating list for f'. And this interval (a, b) can contain a root of f if and only if $f(a)f(b) < 0$. If $f(a)f(b) < 0$ then, we add I (refined from $[a, b]$ in line 7) to L if this condition is satisfied. Thus L is a locating list for f. By using Descartes's rule of signs (Theorem 5.3), we know that f has at most $2k - 1$ real roots. Therefore the test in line 6 can succeed at most $2k - 1$ times. Thus L is a list of locating intervals for f of size at most $N + 2k$. □

Now we can use Algorithm 5.6 recursively to compute the integer roots of a given integer (n, k, τ)-nomial.

Now it is easy to see that Algorithm 5.7 proves Theorem 5.11.

Proof of Theorem 5.11. By using Algorithm 5.6, we know that the locating list L for f (in line 5) is of size at most $2k^2$ and is also computed in time $\mathsf{poly}(\tau, \log n, k)$. By Cauchy's

Algorithm 5.7 Compute all the integer roots.

Input: An integer (n, k, τ)-nomial $f(x) \in \mathbb{Z}[x]$.
Output: A set S containing all the integer roots of f.

1: $L_{k-1} \leftarrow \{[0,0]\}$ ▷ L_k is a locating list for the $(k-1)^{\text{th}}$ fractional derivative $f^{[k-1]}$.
2: **for** each $i \in \{0, 1, \ldots, k-2\}$ **do**
3: Using Algorithm 5.6, compute a locating list L_i for the i^{th} fractional derivative $f^{[i]}$ in $(-B, B)$.
4: **end for**
5: $L \leftarrow L_0$ ▷ This is a locating list for f.
6: $S \leftarrow \emptyset$
7: **for** each interval $[a, b] \in L$ **do**
8: **if** $f(a) = 0$ **then**
9: $S \leftarrow S \cup \{a\}$
10: **end if**
11: **if** $f(b) = 0$ **then**
12: $S \leftarrow S \cup \{b\}$
13: **end if**
14: **end for**
15: **return** S

Root bound, we know that all the roots of f lie in $(-B, B)$. Thus for each integer root α of f, there is an interval $[a, b] \in L$ such that $\alpha \in [a, b]$. Also, since we only add refined interval to the L_i's, all the intervals in L are of length at most 1. Therefore all the integer roots of f can only lie on the end points of intervals in L. Therefore it is obvious that the loop in line 7 finds all the integers roots of f. This loop runs in time $\mathsf{poly}(\tau, \log n, k)$. Thus the time complexity of Algorithm 5.7 is $\mathsf{poly}(\tau, \log n, k)$. □

Remark 5.2. Algorithm 5.7 computes all the integer roots of sparse integer polynomials in polynomial time. Lenstra (Jr.) [LJ99] gave an algorithm which can compute all the rational roots of such polynomials in polynomial time.

5.8 Computing the Real Roots of k-nomials

We saw above that the task of computing the integer roots of a k-nomial f can be performed by computing the integer roots of the first fractional derivative $f^{[1]}$. We want to extend the same strategy to compute the real roots as well. To this end, we define the notion of isolating intervals.

Definition 5.8 (Isolating interval list)**.** A list $L = \{(u_i, v_i)\}_{i \in [N]}$ of open intervals with rational end points and satisfying $u_i < v_i < u_{i+1}$, is said to isolate the (real) roots of $f(x) \in \mathbb{R}[x]$ if, for each (real) root α of $f(x)$, there exists an $i \in [N]$ such that $\alpha \in [u_i, v_i]$.

We have seen in Theorem 5.10 that one can not compute a list of isolating intervals for (n, k, τ)-nomials in $\mathsf{poly}(n, k, \tau)$ time. Thus it is reasonable to ask whether in polynomial time one can compute an approximation of all the real roots. To this end, we define:

Definition 5.9 (Weak (L, I)-covering)**.** A *weak* (L, I)*-covering* for f is a list $(I_1, I_2, \ldots, I_t)$ of open disjoint and sorted real intervals that fulfills the following conditions:

1. The width of each interval I_j is at most 2^{-L}.
2. For every real root ξ of f in I, there exists an interval I_j that contains ξ.

We remark that the above Definition 5.9 is quite similar to the definition of locating lists (Definition 5.7) but with an additional parameter L which ensures that the intervals are "small". We use the definition of weak covering to compute a covering defined below in Definition 5.10.

Definition 5.10 ((L, I)-covering)**.** For a polynomial f, an integer $L \in \mathbb{N}$, and an interval $I \subset \mathbb{R}$, we call a list $((\Delta_{r_1}(m_1), \mu_1), (\Delta_{r_2}(m_2), \mu_2), \ldots, (\Delta_{r_t}(m_t), \mu_t))$ an (L, I)*-covering* for f if the following conditions are fulfilled:

1. The disks $\Delta_{r_i}(m_i) \subset \mathbb{C}$ are pairwise disjoint, m_j are real values with $m_1 < \cdots < m_t$, and $r_j \leq 2^{-L}$ for all j.
2. $\Delta_{r_j}(m_j)$ contains exactly μ_j roots of f for all j.
3. For every real root ξ of f in I, there exists some disk $\Delta_{r_j}(m_j)$ that contains ξ.

If $I = \mathbb{R}$, we omit I and just call a (weak) $(L, \mathbb{R})$-covering for f a (weak) L-covering for f. The main result of this chapter is the following Theorem 5.13.

Theorem 5.13. *For any (n, k, τ)-nomial, we can compute an L-covering $\mathcal{L}$ of size $|\mathcal{L}| < 2k$ in time $\tilde{O}(\mathsf{poly}(k, \log n) \cdot (\tau + L))$.*

It is not hard to see that Theorem 5.13 is a true generalization of Theorem 5.11 because we can set $L = 2$ and then check if the disks in L-covering of Theorem 5.13 contain integer roots. Notice that our algorithm computes L-bit approximations of all real roots but might also return (real-valued) L-bit approximations of some non-real roots with a small imaginary part (the centers of the disks $\Delta_{r_i}(m_i)$). Further notice that unless μ_j is odd, we also do not know whether m_j actually approximates a real root, and unless $\mu_j = 1$, we cannot conclude that a disk $\Delta_{r_j}(m_j)$ in an L-covering is isolating for a root of f. Hence, in general, our algorithm does not yield the correct number of distinct real roots. However, if f has only simple roots, we may compute an L-covering for f for $L = 2, 4, 8, \ldots$ until $\mu_j = 1$ for all j. Then, the disks $\Delta_{r_i}(m_i)$ isolate all real roots. This argument implies the following Theorem 5.14.

Theorem 5.14. *Let f be an (n,k,τ)-nomial with only simple real roots, and let σ be the minimal distance between any two (complex) distinct roots of f, i.e., the separation of f. Then, we can compute isolating intervals for all real roots in $\tilde{O}\left(\mathsf{poly}(k,\log n)(\tau+\overline{\log}\left(\frac{1}{\sigma}\right))\right)$ bit operations.*

Algorithm of Theorem 5.13 improves upon [Sag14] in several ways. Namely, [Sag14] only applies to integer polynomials, whereas our novel approach applies to polynomials with arbitrary real coefficients. In addition, the running time of the algorithm in [Sag14] does not adapt to the actual separation of the roots, whereas the complexity of our novel approach rather depends on the actual separation than on the worst-case bound of size $2^{-\Theta(n(\tau+\log n))}$ for the separation of an integer polynomial. In the worst case, our method isolates all the real roots of a very sparse integer polynomial, *i.e.*, $k=(\log(n\tau))^{O(1)}$, in time $\tilde{O}(n\tau)$. Thus our bound of $\tilde{O}(n\tau)$ is essentially optimal, as shown in Theorem 5.10.

As an easy implication of Theorem 5.14 and Theorem 5.8, the following result is easy to prove.

Corollary 5.6. Let $f(x)=a_1x^{e_1}+a_2x^{e_2}+a_3$ *be an integer trinomial satisfying the condition:* $\log\max(e_1,e_2,|a_1|,|a_2|,|a_3|)\leq\tau$ *and* $e_2\leq e_1=n$. *Then we can isolate all the real roots of* $f(x)$ *in* $\tilde{O}\left(\mathsf{poly}(k,\log n)\cdot\tau^3\right)$ *bit operations.*

Proof. The claim follows immediately by applying Theorem 5.14 and Theorem 5.8, which essentially proves that $\sigma_f\geq 2^{-2^{68}\tau^3}$. More specifically, we can compute an L-covering (by using Theorem 5.13) with $L>2^{69}\tau^3$. This also covers the case of $f(x)$ having double roots. □

5.9 Overview of the Algorithm

Before we go into detail, we give a brief overview of our algorithm, where we omit the technical details. We first remark that the problem of computing an $(L,[1,\infty))$-covering can be reduced to the problem of computing an $(L,[0,1])$-covering (in fact, we are computing an $(L,[0,1+\frac{1}{n}])$-covering but this is only due to some technical reasons) by means of the coordinate transformation $x\mapsto\frac{1}{x}$ followed by multiplication with x^n. We may also reduce the problem of computing an $(L,(-\infty,0])$-covering of f to the problem of computing an $(L,[0,\infty))$-covering by means of the coordinate transformation $x\mapsto -x$. Hence, we are eventually left with merging $(L,[0,1])$-coverings for the polynomials f, $x^n\cdot f(\frac{1}{x})$, $f(-x)$, and $x^n\cdot f(-\frac{1}{x})$ in a suitable manner. We give details for this step in Section 5.14. Notice that the considered coordinate transformation preserves the sparseness of the input polynomial, hence we may concentrate on the problem of computing an $(L,[0,1])$-covering for f only. For this, we first compute a weak $(L,[0,1])$-covering of f, which is achieved by recursively computing weak $(L,[0,1])$-coverings of the fractional derivatives of f.

The general idea of recursively computing the real roots of f from the real roots of its fractional derivatives has already been considered in previous work; e.g. [LJ99; Sag14; GG12; CL76; Cos+05; Pan+07; RY05; CKS99]. We already explained this idea in proving Theorem 5.11, where we computed a locating list for f if we are given a locating list for the first fractional derivative $f^{[1]}$. Let us recall the essential argument we used. Given a weak $(L, [0,1])$-covering $(I'_1, I'_2, \ldots, I'_t)$ for $f^{[1]}$, we already know that in between two consecutive intervals $I_j = (a, b)$ and $I_{j+1} = (c, d)$, the polynomial f is monotone, and thus there can be at most one real root in between b and c, which then must be simple. In order to check for the existence of such a root, it suffices to check whether f changes signs at the points b and c. In case of a sign change, we may then refine the interval (b, c), which is known to be isolating for a real root of f, to a width less than 2^{-L}. If we proceed in this way for all intervals in between two consecutive intervals as well as with the leftmost interval, whose endpoints are 0 and the left endpoint of I'_1, and the rightmost interval, whose endpoints are the right endpoint of I'_t and 1[2], then we obtain a set of intervals I''_j of size at most 2^{-L} that cover all real roots of f that are contained in $[0,1]$ but in none of the intervals I'_j. Hence, the union of the intervals I'_j and I''_j constitutes a weak $(L, [0,1])$-covering for f. This shows how to compute a weak $(L, [0,1])$-covering for f from recursively computing weak $(L, [0,1])$-coverings for its fractional derivatives.

We remark that, in this simplistic description, we have omitted several key problems one faces when formalizing the algorithm: Evaluating the sign of a polynomial f at given points b, c may require a very high precision, which should be avoided to ensure a polynomial bit complexity. In addition, we need an efficient refinement method that uses only a polynomial number of iterations. For the latter problem, we use a slightly modified variant of the algorithm from [Sag14; SM16]. For the computation of the sign of f (and its higher order fractional derivatives) at certain points, we consider an approach that allows us to slightly perturb the evaluation points such that the absolute value of each of the considered polynomials does not become too small. One major contribution of this paper, when compared to our previous work [Sag14], is to show that this can be done in way such that the precision always stays polynomial in $\log n$, k, τ, and L.

In the second phase, we derive an $(L, [0,1])$-covering from a weak $(L', [0,1])$-covering, where L' has been chosen sufficiently large. A straight forward approach would be to use a method for computing the number of roots in the *one-circle region* $\Delta(I) = \Delta_r(m)$ of each interval I in the weak $(L', [0,1])$-covering. Here, $\Delta(I)$ is defined as the disk centered at the midpoint $m = m(I)$ of I and passing through the endpoints of the interval. In the literature, several methods have been proposed to count the number of roots in a disk in complex space. Unfortunately, these algorithm are not sparsity aware, which rules out a straight-forward application of them. Recent work [Bec+18] introduces the so-called T_*-test, a method for root counting based on Pellet's Theorem.

2 For technical reasons, we will indeed consider slight perturbations of 0 and 1 in our algorithm.

The method only needs to compute approximations of the coefficients of the polynomial $f(m + r \cdot x)$, however, we cannot afford to compute all coefficients. Fortunately, in our situation, only the first k^2 coefficients are actually needed to determine the outcome of the test. In order to guarantee success of the test, it may further be necessary to merge some of the intervals in the weak covering and to consider disks that are larger than the one-circle regions of the merged intervals. This explains why we need a weak $(L', [0,1])$-covering with a sufficiently large $L' > L$.

5.10 Polynomial arithmetic

Our algorithm only needs to perform basic operations on polynomials. In particular, we need to evaluate the sign of a given sparse polynomial at some points x. As we already mentioned in the overview of our algorithm, the complexity of this operation becomes too high if the value of the polynomial at a given point x is almost zero as then one needs to perform computations with a very high precision. Also, exact evaluation of a sparse polynomial at a rational point (even of small bit-size) is expensive as the output has bit-size linear in n. Instead, we consider approximate evaluation, which allows us to evaluate any (n, k, τ)-nomial f at an arbitrary point $x \in (0, 1 + \frac{1}{n})$ to an absolute error less than 2^{-L} in a time that is polynomial[3] in $\log n$, k, τ, and L. More precisely, we prove the following Lemma 5.14.

Lemma 5.14. *Let $f \in \mathbb{R}[x]$ be an (n, k, τ)-nomial, c be a positive real number, and L a non-negative integer. Then, we can compute an L-bit approximation λ of $f(c)$ ($|\lambda - f(c)| < 2^{-L}$) in a number of bit operations bounded by*

$$\tilde{O}((k + \log n) \cdot (L + n\overline{\log}(c) + \log n + \tau + k)).$$

Proof. In essence, we follow the same approach as in [KS15]. That is, for a fixed non-negative integer K, we perform each occurring operation $\circ \in \{+, \cdot\}$ (either addition or multiplication) with fixed precision K. More precisely, the input is initially rounded after the K^{th} bit after the binary point. Then, in each of the following steps, we replace each exact operation $\circ$ between two numbers a and b by a corresponding approximate operation $\tilde{\circ}$, where we define $a \tilde{\circ} b$ to be the result obtained by rounding $a \circ b$ after the K^{th} bit after the binary point. Suppose that we have computed approximations $\tilde{a} = a + \varepsilon_1$ and $\tilde{b} = b + \varepsilon_2$ of two intermediate results a and b, where we assume that $\varepsilon \stackrel{\text{def}}{=\!=} \max(|\varepsilon_1|, |\varepsilon_2|, 2^{-K}) < 1$. Then, we have:

$$|a \cdot b - \tilde{a} \cdot \tilde{b}| < |a| \cdot |\varepsilon_1| + |b| \cdot |\varepsilon_2| + |\varepsilon_1 \varepsilon_2| + 2^{-K} < 4 \cdot \varepsilon \cdot \max(1, |a|, |b|)$$

and

$$|a + b - (\tilde{a} + \tilde{b})| < |\varepsilon_1| + |\varepsilon_2| + 2^{-K}.$$

3 Notice that, for $c \in (0, 1 + \frac{1}{n^{O(1)}})$, we may omit the term $n\overline{\log}(c)$ in the bound stated in Lemma 5.14.

Hence, when evaluating one term $f_i \cdot x^{e_i}$ of f at the point $x = c$ with absolute precision

$$K > L + \log k + 1 + \tau + (2\log n + 1) \cdot (n \cdot \overline{\log}(c) + 2)$$

via repeated squaring, we induce a total error ε_i for the computation of $f_i \cdot c^{e_i}$ of size less than

$$2^{\tau} c^{n(2\log n+1)} \cdot 4^{2\log n+1} \cdot 2^{-K-L} < (2k)^{-1} \cdot 2^{-L}$$

as there are at most $2 \cdot \log n + 1$ multiplications, and each (exact) intermediate result has absolute value bounded by $\max(2^{\tau}, c^n)$. When eventually summing up the approximations of all terms $f_i \cdot x^{e_i}$, we thus induce an error of size less than $\sum_i \varepsilon_i + k \cdot 2^{-K} < 2^{-L}$ for the computation of the final result. The bound on the bit complexity follows from the fact that we need $O(k + \log n)$ arithmetic operations on integers of bit-size

$$O(K + \tau + \log k + n\overline{\log}(c)) = O(K)$$

and each such operation uses $\tilde{O}(K)$ bit operations. □

We already mentioned that evaluating the sign of a polynomial f at a point x might be costly if $f(x)$ has a small absolute value. In order to avoid such undesired computations, we first perturb x in a suitable manner. That is, instead of evaluating the sign of f at x, we evaluate its sign at a nearby point, where f becomes large enough. This can be done in a way such that the actual behavior of the algorithm does not change. We will call such points "admissible". We remark that this concept was already used in previous work [Sag14; SM16]. Here, we modify the approach to choose an admissible point, where the sign of each fractional derivative of a sparse polynomial f can be evaluated in polynomial time.

Definition 5.11 (Multi-point)**.** For $m \in \mathbb{R}, \delta \in \mathbb{R}_+$ and $t \in \mathbb{N}$, the *multi-point* $m[t;\delta]$ is defined as:

$$m[t;\delta] \overset{\text{def}}{=\!=} \{m_i \overset{\text{def}}{=\!=} m + (i-t) \cdot \delta; i = 0, 1, \ldots, 2t\}.$$

Definition 5.12 (Admissible point)**.** Let $g : \mathbb{R} \to \mathbb{R}$ be a function, a point $m^* \in m[t;\delta]$ is said to be $(g, m[t;\delta])$-*admissible* if

$$|g(m^*)| \geq \frac{1}{8} \cdot \max_{x \in m[t;\delta]} |g(x)| .$$

If t and δ (or even m and g) are clear from the context, we simply call a $(g, m[t;\delta])$-admissible point (g, m)-admissible (or just admissible). Since the value of g at an admissible points is "relatively large", we expect that g has no root in a corresponding neighborhood. The following Lemma 5.15 formalizes this intuition.

Lemma 5.15. *Let $m^* \in m[t;\delta]$ be an $(f, m[t;\delta])$-admissible point for an (n, k, τ)-nomial f, with $m \in \mathbb{R}_+$ and $2 \leq k \leq t \leq k^2$. If $\frac{m}{\delta} > 4k^2n^2$, then the disk $\Delta_{\delta \cdot k^{-4k}}(m^*)$ contains no root of f for $n \geq 3$.*

Proof. Let $z_1, z_2, \ldots, z_n$ denote the complex roots of f. We remark that due to the condition $\frac{m}{\delta} > 4k^2n^2$, each disk $\Delta_\delta(m_i)$ is contained in the cone C_n for all $m_i \in m[t;\delta]$.

Since $f(x)$ has at most $k-1$ roots in the cone C_n (see Figure 5.2) and $t \geq k$, there exists a point $m_{i_0} \in m[t;\delta]$ such that $\Delta_\delta(m_{i_0})$ does not contain any of these roots.

By way of contradiction, assume that there is a root z_l of f in the disc of radius $\frac{\delta}{k^{4k}}$ around m^*. We have:

$$\frac{f(m^*)}{f(m_{i_0})} = \prod_{i=1}^{n} \frac{m^* - z_i}{m_{i_0} - z_i}$$

Let d_i be the distance between m_{i_0} and z_i, we know that $d_i \geq \delta$. By using the triangle inequality for the distance between m^* and z_i, one can see that, for the roots $z_i \neq z_l$ that are contained in C_n, we have

$$\left|\frac{m^* - z_i}{m_{i_0} - z_i}\right| \leq 1 + \frac{2\delta t}{d_i} \leq 1 + 2t \leq 1 + 2k^2,$$

whereas $\left|\frac{m^* - z_l}{m_{i_0} - z_l}\right| < k^{-4k}$. Now consider the roots z_j of f that are outside of C_n. We know that $d_j \geq (m - \delta t) \cdot \sin(\frac{\pi}{n+2})$. Since $\frac{2}{\pi}x \leq \sin(x)$, we get that $d_j \geq \frac{2\cdot(m-\delta t)}{n+2}$. By using the inequality $\frac{m}{\delta} > 4k^2n^2$, we get $d_j \geq \frac{2\cdot\delta(4k^2n^2-t)}{(n+2)} \geq \frac{2\cdot\delta k^2(4n^2-1)}{(n+2)} \geq 2\delta k^2 n$. Thus

$$\left|\frac{m^* - z_i}{m_{i_0} - z_i}\right| \leq 1 + \frac{2\delta t}{d_j} \leq 1 + \frac{t}{k^2 n} \leq 1 + \frac{1}{n},$$

Hence:

$$\begin{aligned}
\frac{f(m^*)}{f(m_{i_0})} &\leq \left(1 + \frac{1}{n}\right)^{n-k+1} \cdot (2k^2+1)^{k-2} \cdot \frac{1}{k^{4k}} \\
&< \left(1 + \frac{1}{n}\right)^{n-k+1} \cdot \left(\frac{2k^2+1}{k^4}\right)^{k-2} \cdot \frac{1}{k^8} \\
&\leq \frac{e}{k^8} \\
&< \frac{1}{8}.
\end{aligned}$$

But this contradicts the fact that m^* is an admissible point. □

Note that we defined the notion of an admissible point for general functions and not just for polynomials. This is because during the computation of a weak L-covering of f, we aim to compute admissible points where the values of all fractional derivatives are large. For this, we need the following definition.

Definition 5.13. Let $\mathcal{G} = (g_1, g_2, \ldots, g_t)$ be a tuple of t functions $g_i : \mathbb{R} \to \mathbb{R}$. Then, $M_{\mathcal{G}}(x)$ is defined as follows:

$$M_{\mathcal{G}}(x) \stackrel{\text{def}}{=\!=} \min(|g_1(x)|, |g_2(x)|, \ldots, |g_t(x)|).$$

For a fixed real x, we call $\tilde{\mathcal{G}}(x) = (\tilde{g}_1(x), \tilde{g}_2(x), \ldots, \tilde{g}_t(x))$ an *L-approximation of $\mathcal{G}(x)$* if $|\tilde{g}_i(x) - g_i(x)| \leq 2^{-L}$ for all i.

We first show how to compute an admissible point $m^* \in m[t;\delta]$ for $M_{\mathcal{G}}(x)$ under the assumption that we can compute an L-approximation of $\mathcal{G}(x)$ for any $x \in m[t;\delta]$ in time $T(L)$.

Lemma 5.16. *Let $\mathcal{G} = (g_1, g_2, \ldots, g_t)$ be as in Definition 5.13, $m[t;\delta]$ a multi-point and $\lambda := \max_{a \in m[t;\delta]} |M_{\mathcal{G}}(a)|$. Suppose that for any point $m_i \in m[t;\delta]$, we can compute an L-approximation of $\mathcal{G}(m_i)$ in time $T(L)$. Then, we can compute an $(M_{\mathcal{G}}, m[t;\delta])$-admissible point $m^* \in m[t;\delta]$ as well as an integer ℓ^* with*

$$2^{\ell^*-1} \leq |M_{\mathcal{G}}(m^*)| \leq \lambda \leq 2^{\ell^*+1}$$

in time

$$O\left(t \cdot \overline{\log}\left(\overline{\log}\left(\lambda^{-1}\right)\right) \cdot T\left(\overline{\log}\left(\lambda^{-1}\right)\right)\right).$$

Proof. We proceed in the same fashion as in Lemma 8 of [SM16]. For $L = 1, 2, 4, \ldots$, we compute L-approximations $\tilde{\mathcal{G}}_i^L = (\tilde{g}_1^L(m_i), \tilde{g}_2^L(m_i), \ldots, \tilde{g}_t^L(m_i))$ of $\mathcal{G}(m_i)$ for all points $m_i \in m[t;\delta]$ until the following condition is satisfied for at least one m_i:

$$M_i^L \stackrel{\text{def}}{=\!=} \min\left(\left|\tilde{g}_i^L(m_i)\right|, \left|\tilde{g}_2^L(m_i)\right|, \ldots, \left|\tilde{g}_t^L(m_i)\right|\right) \geq 4 \cdot 2^{-L} = 2^{-L+2}.$$

Then, let i_0 be the index such that $M_{i_0}^L$ is maximal among all $M_{i_0}^L$, and let ℓ^* be an integer such that $\left|\ell^* - \log M_{i_0}^L\right| \leq \frac{1}{2}$. We output ℓ^* and $m^* := m_{i_0}$.

Using a similar straight-forward argument as in the proof of Lemma 8 of [SM16] then shows that $2^{\ell^*-1} \leq M_{\mathcal{G}}(m^*) \leq \lambda \leq 2^{\ell^*+1}$. By following this approach, we must succeed for some $L \leq 2\overline{\log}\left(\frac{1}{\lambda}\right)$. Since we double L at most $\log\log\max(\frac{1}{\lambda}, 1)$ many times and since we approximately evaluate the functions g_i at t points, the stated complexity bound follows. □

We now apply the above lemma to $\mathcal{G} \stackrel{\text{def}}{=\!=} \mathcal{D}_f$, the sequence of fractional derivatives of f. Then Lemma 5.14 yields a bound for the bit complexity of computing L-approximations of $\mathcal{D}_f(m_i)$ for all $m_i \in m[t;\delta]$, which directly depends on $\lambda \stackrel{\text{def}}{=\!=} \max_{m_i \in m[t;\delta]} \left|M_{\mathcal{D}_f}(m_i)\right|$.

Corollary 5.7. *Assume that $f(x)$ is an (n,k,τ)-nomial, $m[t;\delta]$ a multi-point and $\lambda \stackrel{def}{=\!=} \max_{m_i \in m[t;\delta]} \left|M_{\mathcal{D}_f}(m_i)\right|$. Further assume that $m[t;\delta] \subset (0,\alpha)$ for some positive real α. Then, we can determine an $(M_{\mathcal{D}_f}, m[t;\delta])$-admissible point m^* and an integer ℓ^* with*

$$2^{\ell*-1} \leq \left|M_{\mathcal{D}_f}(m^*)\right| \leq \lambda \leq 2^{\ell^*+1}$$

using $\tilde{O}\left(t \cdot k \cdot (k+\log n) \cdot \left(\tau + k\log n + n\overline{\log}(\alpha) + \overline{\log}\left(\lambda^{-1}\right)\right)\right)$ many bit operations.

The following bound on λ implies that, for suitably chosen t,m and δ, we can compute m^* in polynomial time.

Lemma 5.17. *Let $f \in \mathbb{R}[x]$ be a (n,k,τ)-nomial with $\tau > 0$ and $f(0) \neq 0$, and let a, r be positive real numbers with $r < a$ such that $(a-r, a+r)$ does not contain any real root of any fractional derivative of $f(x)$. Then,*

$$\left|M_{\mathcal{D}_f}(a)\right| \geq 2^{-k(3\tau+3k\overline{\log}(n)+\overline{\log}\left(\frac{1}{r}\right))}.$$

Proof. First we show that we can assume $n > 1$. The case of $n = 0$ is trivial. If $n = 1$, then we have $f(x) = f_1 x + f_0$ with $2^{-\tau} \leq |f_i| \leq 2^{\tau}$ and $f_1 \neq 0$. The only root of $f(x)$ is $-\frac{f_0}{f_1}$. We have $f^{[1]} = f_1$. Since $(a-r, a+r)$ does not contain any real root of any fractional derivative of $f(x)$, we get that $\left|a + \frac{f_0}{f_1}\right| > r$. Therefore $f(a) = f_1 a + f_0 = f_1\left(a + \frac{f_0}{f_1}\right)$. Hence $|f(a)| > r2^{-\tau} \geq 2^{-\tau+\overline{\log}\left(\frac{1}{r}\right)}$. In particular, we have $\left|M_{\mathcal{D}_f}(a)\right| \geq 2^{-k(3\tau+3k\log n+\overline{\log}\left(\frac{1}{r}\right))}$. Thus the condition $n > 1$ can be assumed without loss of generality.

We prove this claim by induction on the sparsity of f. Our induction hypothesis is: if $(a-r, a+r)$ does not contain any real root of any fractional derivative of $f(x)$ for an (n,i,τ)-nomial $f(x)$ with $f(0) \neq 0$ then

$$\left|M_{\mathcal{D}_f}(a)\right| \geq 2^{-i(3\tau+3i\log n+\overline{\log}\left(\frac{1}{r}\right)+i)}.$$

Let f be an $(n,i+1,\tau)$-nomial with $f(0) \neq 0$. We have that $f = a + x^j g$ with $j \geq 1$, here g is some i-nomial and also $a \neq 0, g(0) \neq 0$. We have $f' = jx^{j-1}g + x^j g'$ and $f^{[1]} = jg + xg'$. Since $(a-r, a+r)$ does not contain any root of any fractional derivative of $f(x)$, it follows that f is monotone in $(a-r, a+r)$. We assume that f is positive in $(a-r, a+r)$, otherwise we apply the analysis below to $-f$.

Now we have the following two cases.

Case 1. First case is when f is increasing in $(a-r, a+r)$. In this case, mean value theorem guarantees the existence of a $t \in \left(a-\frac{r}{2}, a\right)$ such that the following inequality is true.

$$\begin{aligned} f(a) &> f(a) - f\left(a-\frac{r}{2}\right) \\ &= \frac{r}{2} \cdot f'(t) = \frac{r}{2} \cdot t^{j-1} f^{[1]}(t). \end{aligned}$$

Since $(a-r, a+r)$ contains no root of any fractional derivative of f, we get that $\left(t-\frac{r}{2}, t+\frac{r}{2}\right)$ contains no root of any fractional derivative of $f^{[1]}$. Also, $f^{[1]}$ is an $(n, i, \tau+\log n)$-nomial with $f^{[1]}(0) \neq 0$. Therefore we can use the induction hypothesis on $f^{[1]}$ to obtain the following lower bound on $\left|M_{\mathcal{D}_{f^{[1]}}}(t)\right|$.

$$\left|M_{\mathcal{D}_{f^{[1]}}}(t)\right| \geq 2^{-i(3(\tau+\log n)+2i\log n+\overline{\log}\left(\frac{2}{r}\right)+i)}.$$

In particular,

$$\left|f^{[1]}(t)\right| \geq 2^{-i(3(\tau+\log n)+2i\log n+\overline{\log}\left(\frac{2}{r}\right)+i)}.$$

Now we have two sub-cases, in the first sub-case we have that $t^{j-1} \geq 2^{-(3\tau+\log i)}$. In this sub-case, we obtain:

$$\begin{aligned} |f(a)| &> \frac{r}{2} \cdot 2^{-(3\tau+\log i)} \cdot f^{[1]}(t) \\ &\geq 2^{-(i(3\cdot(\tau+\log n)+2i\log n+\overline{\log}\left(\frac{2}{r}\right)+i)+\log(\frac{2}{r})+3\tau+\log i)}. \end{aligned} \tag{5.5}$$

Let us use

$$E \stackrel{\text{def}}{=\!=} (i(3\cdot(\tau+\log n)+2i\log n+\overline{\log}\left(\frac{2}{r}\right)+i)+\log(\frac{2}{r})+3\tau+\log i)$$

to denote the negation of exponent in Equation (5.5). To complete the induction step, we need to prove that

$$E \leq (i+1)(3\tau+2(i+1)\log n+\overline{\log}\left(\frac{1}{r}\right)+i+1).$$

We use the fact that

$$\log(\frac{2}{r}) \leq \overline{\log}\left(\frac{2}{r}\right)$$

and

$$\overline{\log}\left(\frac{2}{r}\right) \leq 1+\overline{\log}\left(\frac{1}{r}\right).$$

This yields the following upper bound for E.

$$\begin{aligned}
E &\leq 3(i+1)\tau + 3i\log n + 2i^2\log n + i\overline{\log}\left(\frac{1}{r}\right) + i + \overline{\log}\left(\frac{1}{r}\right) + 1 + \log i + i^2 \\
&\leq 3(i+1)\tau + (2i^2+3i+1)\log n + (i+1)\overline{\log}\left(\frac{1}{r}\right) + i^2 + i + 1 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(Because } i \leq n\text{)} \\
&\leq 3(i+1)\tau + 2(i+1)^2\log n + (i+1)\overline{\log}\left(\frac{1}{r}\right) + (i+1)^2 \\
&\leq (i+1)(3\tau + 2(i+1)\log n + \overline{\log}\left(\frac{1}{r}\right) + i + 1).
\end{aligned}$$

This implies that

$$|f(a)| > 2^{-(i+1)(3\tau+2(i+1)\log n+\overline{\log}\left(\frac{1}{r}\right)+i+1)}.$$

Now we look at the sub-case when $t^{j-1} < 2^{-(3\tau+\log i)}$. In this case, $t < 1$. Therefore $|g(t)| \leq 2^\tau i = 2^{\tau+\log i}$. Hence

$$\left|t^j g(t)\right| \leq \left|t^{j-1} g(t)\right| < 2^{-(3\tau+\log i)} 2^{\tau+\log i} \leq 2^{-2\tau}.$$

Thus $|f(a)| > |f(t)| = \left|a + t^j g(t)\right| \geq 2^{-\tau+1}$. By using the induction hypothesis on $f^{[1]}$, we obtain that

$$\left|M_{\mathcal{D}_f}(a)\right| \geq 2^{-(i+1)(3\tau+2(i+1)\log n+\overline{\log}\left(\frac{1}{r}\right)+i+1)}.$$

Case 2. Now consider the case when f is decreasing in the interval $(a-r, a+r)$. By a similar argument as above, we get that there exists a $t \in \left(a, a+\frac{r}{2}\right)$ such that the following inequality is true.

$$\begin{aligned}
f(a) &> f(a+\frac{r}{2}) > f(a+\frac{r}{2}) - f\,(a) \\
&= \frac{r}{2}\cdot f'(t) = \frac{r}{2}\cdot t^{j-1} f^{[1]}(t).
\end{aligned}$$

Since $(a-r, a+r)$ contains no root of any fractional derivative of f, we get that $\left(t-\frac{r}{2}, t+\frac{r}{2}\right)$ contains no root of any fractional derivative of $f^{[1]}$. Now we follow exactly the same analysis as we applied above in the case when f was monotonically increasing in the interval $(a-r, a+r)$. This yields:

$$\left|M_{\mathcal{D}_f}(a)\right| \geq 2^{-(i+1)(3\tau+2(i+1)\log n+\overline{\log}\left(\frac{1}{r}\right)+i+1)}.$$

□

Since $n > 1$, we have that $k \log n \geq k$. Thus the claimed lower bound on $\left|M_{\mathcal{D}_f}(a)\right|$ follows.

Combining the above lemma and Corollary 5.7 now yields

Theorem 5.15. *Let f be a (n, k, τ)-nomial and let $m[t; \delta]$ be a multi-point with $t \geq k^2$ and $m[t; \delta] \subset (0, \alpha)$ for some for some real number α. Then, we can compute an $(M_{\mathcal{D}_f}, m[t; \delta])$-admissible point m^* using*

$$\tilde{O}(t \cdot k^2 \cdot (k + \log n) \cdot (k \log n + \tau + \overline{\log}\left(\delta^{-1}\right) + n\overline{\log}\left(\alpha\right)))$$

bit operations.

Proof. Since each fractional derivative of f has at most $k - 1$ positive real roots and since $t \geq k^2$, there exists an $a \in m[t; \delta]$ such that $(a - \delta/2, a + \delta/2)$ does not contain any real root of any of fractional derivative. Hence, Lemma 5.17 implies that $\lambda := \max_{x \in m[t;\delta]} \left|M_{\mathcal{D}_f}(x)\right| \geq \left|M_{\mathcal{D}_f}(a)\right|$ is lower bounded by $2^{-O(k(k \log n + \tau + \overline{\log}(\delta^{-1}))}$. Corollary 5.7 then yields the claimed bound on the running time. □

Now we prove Theorem 5.16, which essentially states that if a small ball around a real number x_0 does not contain any root of $f(x)$ then $|f(x_0)|$ is "large". First we prove the following Lemma 5.18.

Lemma 5.18. *Let $f = \sum_{i=0}^{n} f_i x^i \in \mathbb{C}[x]$ be any polynomial. If $a, b \in \mathbb{R}^+$ are such that $|a - b| \leq \delta$ and $\Delta_{2\delta n}(\frac{a+b}{2})$ does not contain any root of f then $\left|\frac{f(a)}{f(b)}\right| \in [\frac{1}{e}, e]$.*

Proof. Suppose $z_1, z_2, \ldots, z_n \in \mathbb{C}$ are the roots of f. We have that

$$\left|\frac{f(a)}{f(b)}\right| = \prod_{i=1}^{n} \left|\frac{a - z_i}{b - z_i}\right|.$$

Since $\Delta_{2\delta n}(\frac{a+b}{2})$ does not contain any root of f, for all z_i we have that $\left|\frac{a-z_i}{b-z_i}\right| \geq \frac{2\delta n - \frac{\delta}{2}}{2\delta n + \frac{\delta}{2}} = 1 - \frac{1}{2n + \frac{1}{2}}$. Thus $\left|\frac{f(a)}{f(b)}\right| \geq \left(1 - \frac{1}{2n+\frac{1}{2}}\right)^{\frac{1}{n}} \geq \frac{1}{e}$. Similarly, we can show that $\left|\frac{f(b)}{f(a)}\right| \geq \frac{1}{e}$. Thus $\left|\frac{f(a)}{f(b)}\right| \in [\frac{1}{e}, e]$. □

The following Theorem 5.16 shows that if there are no roots of an (n, k, τ)-nomial $f(x)$ "near" a point x_0 then $|f(x_0)|$ is "large".

Theorem 5.16. *Let $f \in \mathbb{R}[x]$ be an (n, k, τ)-nomial with a non-zero constant term, and let x_0, r be positive real numbers such that $\Delta_r(x_0)$ contains no root of f then*

$$|f(x_0)| = 2^{-O\left(k(k \log n + \tau + \overline{\log}\left(\frac{1}{r}\right)\right)}.$$

Proof. Consider the multi-point $x_0[k^2;\delta]$ for $\delta = \frac{r}{8nk^2}$. There exists a point $x^* \in x_0[k^2;\delta]$ such that $(x^* - \delta, x^* + \delta)$ does not contain any root of any fractional derivative of f. Thus, by using Lemma 5.17, we know that,

$$|f(x^*)| \geq \left|M_{\mathcal{D}_f}(x^*)\right| \geq 2^{-k\cdot\left(3k\log n+3\tau+\overline{\log}\left(\frac{1}{\delta}\right)\right)}. \tag{5.6}$$

Also, the condition of Lemma 5.18 are fulfilled for $a = x^*$ and $b = x_0$. Hence, we obtain that $|f(x_0)| \geq \frac{1}{e}\cdot|f(x^*)|$, and thus

$$|f(x_0)| = 2^{-O\left(k\cdot\left(k\log n+\tau+\overline{\log}\left(\frac{1}{\delta}\right)\right)\right)}. \tag{5.7}$$

By using $\delta = \frac{r}{8nk^2}$, Equation (5.7) can be written as $|f(x_0)| = 2^{-O\left(k\left(k\log n+\tau+\overline{\log}\left(\frac{1}{r}\right)\right)\right)}$. □

As an application of Lemma 5.17, we prove the following Theorem 5.17 which lower bounds $\left|M_{\mathcal{D}_f}(m^*)\right|$ for suitable admissible points m^*.

Theorem 5.17. *Let $f \in \mathbb{R}[x]$ be an (n,k,τ)-nomial and m^* be an $(M_{\mathcal{D}_f}, m[t;\delta])$-admissible point for some $t \geq k^2$. Then, we have that:*

$$\left|M_{\mathcal{D}_f}(m^*)\right| \geq 2^{-\left(k\left(3\tau+3k\log n+\overline{\log}\left(\frac{1}{\delta}\right)\right)+3\right)}.$$

Proof. Since each fractional derivative of f has at most $k-1$ positive real roots and since $t \geq k^2$, there exists an $a \in m[t;\delta]$ such that $(a-\delta/2, a+\delta/2)$ does not contain any real root of any of fractional derivative. Hence, Lemma 5.17 implies that $\left|M_{\mathcal{D}_f}(a)\right| \geq 2^{-k\left(k\log n+\tau+\overline{\log}\left(\delta^{-1}\right)\right)}$. By the definition of admissible points, we get that:

$$\left|M_{\mathcal{D}_f}(m^*)\right| \geq 2^{-\left(k\left(3\tau+3k\log n+\overline{\log}\left(\frac{1}{\delta}\right)\right)+3\right)}.$$

□

5.11 Refinement

A crucial subroutine of our overall algorithm is an efficient method for refining an interval $I_0 = (a_0, b_0) \subset \mathbb{R}_+$ (in what follows, it is assumed that $\overline{\log}\,(a_0, b_0) = O(\tau)$) that is known to be isolating for a simple real root of an (n,k,τ)-nomial f. It is assumed that the algorithm receives the sign of f at the endpoints of I_0 as additional input. For the refinement, we consider the algorithm Refine from Section 5 in [SM16] (see also Section 3 in [Sag14]), however, we make a single (minor) modification. As the argument from [SM16] directly applies, we only state the main results and refer the reader to [SM16] for details.

REFINE recursively refines I_0 to a size less than 2^{-L} using a trial and error approach that combines Newton iteration and bisection. This was modified to work for k-nomials in [Sag14]. The refinement routine in [Sag14] was called NEWREFINE. In each iteration, NEWREFINE computes $(f, m[\lceil k/2 \rceil; \delta])$-admissible points m^* for a constant number of points $m \in I$ and a corresponding δ of size $2^{-O(\tau+\log n+L)}$. In addition, f, f' are evaluated at these admissible points to an absolute precision that is bounded by $O\left(\overline{\log}\left(|f(m^*)|^{-1}\right) + \log n + L + \tau\right)$ and only f, f' need to be evaluated. Each endpoint of the interval returned by NEWREFINE is then either one of the admissible points computed in a previous iteration or one of the endpoints of I_0.

We now propose the following modification of NEWREFINE, which we denote NEWREFINE*(Algorithm 5.11): Whenever NEWREFINE asks for an $(f, m[\lceil k/2 \rceil; \delta])$-admissible point m^*, we compute an $(M_{\mathcal{D}_f}, m[k^2; \delta'])$-admissible point m^*, with $\delta' \approx \frac{\delta}{n}$, instead (see Algorithm 5.8 for a more precise description). We work with these admissible points to guarantee that all occurring endpoints are admissible. As a consequence, computing the signs of all fractional derivatives at the relevant end points is efficient!

To formally define NEWREFINE*, we first need a straight forward modification (Algorithm 5.8) of the "Newton-Test_signat" test proposed in [Sag14]. We call this modification NEWTONTEST*. For missing details, we refer the reader to [SM16] for more details.

We say that Algorithm 5.8 fails if the output pair $(I', N_{I'})$ is the same as the input pair (I, N_I). Otherwise, we say that Algorithm 5.8 succeeds. Now the following Lemma 5.19 is easy to verify.

Lemma 5.19. *If* NEWTONTEST* *(Algorithm 5.8) succeeds, then it returns a pair* $(I', N_{I'})$ *with* $N_{I'} = N_I^2$ *and* $\frac{w(I)}{8N_I} \leq w(I') \leq \frac{w(I)}{N_I}$.

It can happen that the NEWTONTEST* fails. Sufficient conditions for the success of NEWTONTEST* were derived in [SM16]. It is shown in [SM16] that if the roots in I are "well-separated" from other roots of f then the NEWTONTEST* succeeds. More specifically, it was shown in [SM16] that it succeeds if there exists a sub-interval J of I with $w(J) \leq \frac{2^{-13} \cdot w(I)}{N_I}$ such that the one circle region $\Delta(J)$ of J contains all the roots contained in one circle region $\Delta(I)$ of I, and $\Delta_{2^{10} n N_I w(I)}(m(I))$ contains no further roots of f. For an exact statement, see Lemma 23 in [SM16]. One way NEWTONTEST* can fail is that if the there is some cluster of roots in $\Delta(I)$ near one of the end points of I, in this case the test BOUNDARYTEST* succeeds. So if NEWTONTEST* fails, we try the test called BOUNDARYTEST* (Algorithm 5.9). This is a straight forward modification of the "Boundary-Test_signat" test defined in [Sag14] (see also [SM16]).

It is shown in [SM16] that if all the roots of f contained in $\Delta(I)$ are "close" to one of the end points of I then BOUNDARYTEST* succeeds (see "Algorithm: Boundary-Test" in [SM16]).

Algorithm 5.8 NEWTONTEST*

Input: An (n,k,τ)-nomial f, a non-negative integer t with $t < k$, a pair (I, N_I) , here $I = (a,b)$ is an interval with dyadic end points which isolates a real root α of $f^{[t]}$. Also, $N_I = 2^{2^{n_I}}$ for some $n_I \in \mathbb{N}$.

Output: A pair $(I', N_{I'})$, here I' is a sub-interval of I, $\alpha \in I'$ and $N_{I'} \geq N_I$.

1: $\xi_j \leftarrow a + \frac{j}{4} \cdot w(I)$ for $j \in \{1,2,3\}$, $\epsilon \leftarrow 2^{-\lceil 5+2\log n \rceil}$ and $g \leftarrow f^{[t]}$.
2: Compute points ξ_j^* for $j \in \{1,2,3\}$, where ξ_j^* is an $(M_{\mathcal{D}_f}, \xi_j[k^2; \epsilon \cdot w(I)])$-admissible point.
3: **for** all pairs $(j_1, j_2) \in \{(1,2),(1,3),(2,3)\}$ **do**
4: $v_{j_1} \overset{\text{def}}{=\!=} \frac{g(\xi_{j_1}^*)}{g'(\xi_{j_1}^*)}$ and $v_{j_2} \overset{\text{def}}{=\!=} \frac{g(\xi_{j_2}^*)}{g'(\xi_{j_2}^*)}$.
5: Define the Condition1 as: $\Big(\min\left(|v_{j_1}|, |v_{j_2}|\right) > w(I)$ or $|v_{j_1} - v_{j_2}| < \frac{w(I)}{4n})\Big)$.
6: Define the Condition2 as: $\Big(\max\left(|v_{j_1}|, |v_{j_2}|\right) < 2 \cdot w(I)$ and $|v_{j_1} - v_{j_2}| > \frac{w(I)}{8n})\Big)$.
7: Compute a sufficiently good approximation of v_{j_1} and v_{j_2} such that at least one of the conditions (Condition1 or Condition2) can be verified.
8: **if** Condition1 can be verified **then**
9: **break** ▷ Discard the pair (j_1, j_2).
10: **end if**
11: **if** Condition2 can be verified **then**
12: Define $\lambda_{j_1,j_2} \overset{\text{def}}{=\!=} \xi_{j_1}^* + \frac{\xi_{j_2}^* - \xi_{j_1}^*}{v_{j_1} - v_{j_2}} \cdot v_{j_1}$.
13: Compute approximation $\tilde{\lambda}_{j_1,j_2}$of λ_{j_1,j_2} such that $|\lambda_{j_1,j_2} - \tilde{\lambda}_{j_1,j_2}| \leq \frac{1}{32N_I}$.
14: **if** $\tilde{\lambda}_{j_1,j_2} \notin [a,b]$ **then**
15: **break** ▷ Discard the pair (j_1, j_2).
16: **else**
17: Compute $\ell_{j_1,j_2} \overset{\text{def}}{=\!=} \lfloor \frac{4N_I \cdot (\tilde{\lambda}_{j_1,j_2} - a)}{w(I)} \rfloor$.
18: $a_{j_1,j_2} \leftarrow a + \max(0, \ell_{j_1,j_2} - 1) \cdot \frac{w(I)}{4N_i}$ and $b_{j_1,j_2} \leftarrow a + \min(4N_I + \ell_{j_1,j_2} + 2)$
19: **if** $a_{j_1,j_2} = a$ **then**
20: $a_{j_1,j_2}^* \leftarrow a$
21: **else**
22: Compute an $(M_{\mathcal{D}_f}, a_{j_1,j_2}[k^2; \epsilon \cdot \frac{w(I)}{N_I}])$-admissible point c_{j_1,j_2}^*.
23: $a_{j_1,j_2}^* \leftarrow c_{j_1,j_2}^*$
24: **end if**
25: Repeat line 19 to line 24 for b_{j_1,j_2}. ▷ We check if $b_{j_1,j_2} = b$
26: Compute the sign of $g(a_{j_1,j_2}^*)$ and $g(b_{j_1,j_2}^*)$.
27: **if** $g(a_{j_1,j_2}^*) \cdot g(b_{j_1,j_2}^*) < 0$ **then**
28: $I' \leftarrow (a_{j_1,j_2}^*, b_{j_1,j_2}^*)$
29: **return** (I', N_I^2)
30: **else**
31: **break** ▷ Discard the pair (j_1, j_2).
32: **end if**
33: **end if**
34: **end if**
35: **end for**
36: **return** (I, N_I)

Algorithm 5.9 BOUNDARYTEST*

Input: An (n, k, τ)-nomial f, a non-negative integer t with $t < k$, a pair (I, N_I) , here $I = (a, b)$ is an interval with dyadic end points which isolates a real root α of $f^{[t]}$. Also, $N_I = 2^{2^{n_I}}$ for some $n_I \in \mathbb{N}$.

Output: A pair $(I', N_{I'})$, here I' is a sub-interval of I, $\alpha \in I'$ and $N_{I'} \geq N_I$.

1: $m_\ell \leftarrow a + \frac{w(I)}{2N_i}$, $m_r \leftarrow b - \frac{w(I)}{2N_i}$, $\epsilon \leftarrow 2^{-\lceil 2+2\log n \rceil}$ and $g \leftarrow f^{[t]}$.
2: Compute an $(M_{\mathcal{D}_f}, m_\ell[k^2; \epsilon \cdot \frac{w(I)}{N_I}])$-admissible point m_ℓ^* .
3: Compute a $(M_{\mathcal{D}_f}, m_r[k^2; \epsilon \cdot \frac{w(I)}{N_I}])$-admissible point m_r^*.
4: Compute the signs of $g(a), g(m_\ell^*), g(m_r^*)$ and $g(b)$.
5: **if** $g(a) \cdot g(m_\ell^*) < 0$ **then**
6: **return** $((a, m_\ell^*), N_I^2)$
7: **else if** $g(b) \cdot g(m_r^*) < 0$ **then**
8: **return** $((m_r^*, b), N_I^2)$
9: **end if**
10: **return** (I, N_I)

Similar to that of Algorithm 5.8, we say that Algorithm 5.9 fails if the output pair $(I', N_{I'})$ is same as the input pair (I, N_I). Otherwise, we say that Algorithm 5.9 succeeds. Now, the following Lemma 5.20 is easy to verify.

Lemma 5.20. *If* BOUNDARYTEST* *(Algorithm 5.9) succeeds, then it returns a pair* $(I', N_{I'})$ *with* $N_{I'} = N_I^2$ *and* $\frac{w(I)}{4N_I} \leq w(I') \leq \frac{w(I)}{N_I}$.

It might happen that both, the NEWTONTEST* and the BOUNDARYTEST* fail (they can not fail too many times), in this case we fall back to the usual bisection method. Algorithm BISECTIONTEST* (Algorithm 5.10) formalizes this step.

Now, we propose the claimed modification NEWREFINE* of NEWREFINE in [Sag14].

Then, a similar argument[4] as used in Lemma 6 and Theorem 7 of [Sag14] yields the following Theorem 5.18.

Theorem 5.18. *For refining an interval* $I = (a, b)$ *with* $\overline{\log}(a, b) = O(\tau)$ *to a size less than* 2^{-L}, *the algorithm* NEWREFINE* *(Algorithm 5.11) needs* $O(k \cdot (\log n + \log(\tau + L)))$ *iterations. In each iteration, we need to compute a constant number of* $(M_{\mathcal{D}_f}, m[k^2; \delta'])$*-admissible points* m^*, *with* $m[k^2; \delta'] \subset I_0$ *and* $\delta' = 2^{-O(\tau + \log n + L)}$. *In addition, the polynomials* $f^{[t]}$ *and* $(f^{[t]})'$ *have to be evaluated at* m^* *to an absolute precision bounded by* $O\left(\overline{\log}\left(|f(m^*)|^{-1}\right) + \log n + L + \tau\right)$.

Proof Sketch. Let $I = I_0 \supset I_1 \supset \cdots \supset I_s$ be the chain of intervals produced by NEWREFINE*. Let $s_{\max}$ be the length of the longest continuous sub-chain in $I_0 \supset I_1 \supset$

4 The argument in Sagraloff [Sag14] only uses that, in each iteration, we choose an arbitrary point $m^* \in [m - \lceil k/2 \rceil \cdot \delta, m + \lceil k/2 \rceil \cdot \delta]$.

Algorithm 5.10 BISECTIONTEST*

Input: An (n, k, τ)-nomial f, a non-negative integer t with $t < k$, a pair (I, N_I) , here $I = (a, b)$ is an interval with dyadic end points which isolates a real root α of $f^{[t]}$. Also, $N_I = 2^{2^{n_I}}$ for some $n_I \in \mathbb{N}$.

Output: A pair $(I', N_{I'})$, here I' is a sub-interval of I, $\alpha \in I'$ and $N_{I'} \geq N_I$.

1: $\epsilon \leftarrow 2^{-\lceil 2+2\log n \rceil}$.
2: $g \leftarrow f^{[t]}$.
3: Compute admissible point m^*, where m^* is an $(M_{\mathcal{D}_f}, m(I)[k^2; \epsilon \cdot w(I)])$-admissible point.
4: Compute the signs of $g(a), g(m^*), g(m_r^*)$ and $g(b)$.
5: $N_{I'} \leftarrow \max\left(4, \sqrt{N_I}\right)$
6: **if** $g(a) \cdot g(m^*) < 0$ **then**
7: **return** $((a, m^*), N_{I'})$
8: **else**
9: **return** $((m^*, b), N_{I'})$
10: **end if**

Algorithm 5.11 NEWREFINE*

Input: An (n, k, τ)-nomial f, a non-negative integer t with $t < k$, an interval I and a positive integer L. Here I is an interval with dyadic end points which isolates a real root α of $f^{[t]}$.

Output: A sub-interval I' of I with $|I| \leq 2^{-L}$ and $\alpha \in I'$.

1: $N_{I'} \leftarrow 4$.
2: $I' \leftarrow I$.
3: **while** $|I'| > 2^{-L}$ **do**
4: $(I_1, N_{I_1}) \leftarrow$ Output of NEWTONTEST* (Algorithm 5.8) on $(I', N_{I'})$.
5: **if** $N_{I_1} = N_{I'}^2$ **then**
6: $(I', N_{I'}) \leftarrow (I_1, N_{I_1})$.
7: **continue**
8: **end if**
9: $(I_2, N_{I_2}) \leftarrow$ Output of BOUNDARYTEST* (Algorithm 5.9) on $(I', N_{I'})$.
10: **if** $N_{I_2} = N_{I'}^2$ **then**
11: $(I', N_{I'}) \leftarrow (I_2, N_{I_2})$.
12: **continue**
13: **end if**
14: $(I_3, N_{I_3}) \leftarrow$ Output of BISECTIONTEST* (Algorithm 5.10) on $(I', N_{I'})$.
15: $(I', N_{I'}) \leftarrow (I_3, N_{I_3})$.
16: **end while**
17: **return** I'

$\cdots \supset I_s$ such that the one circle region of all the intervals in this sub-chain contains exactly the same roots of $f^{[t]}$. By using an analogous argument as in Lemma 26 in [SM16], (see also Lemma 6 in [Sag14]), we get that $s_{\max} = O(\log n + \log(\tau + L))$. Let $v_0 \overset{\text{def}}{=\!=} \text{var}(f^{[t]}, I_0)$. The main task in Theorem 7 of [Sag14] is to show that $s = O(v_0 \cdot (\log n + \log(\tau + L)))$. To this end, [Sag14] divides the chain $I_0 \supset I_1 \supset \cdots \supset I_s$ into two parts. Let j_0 be the smallest index such that $\Delta(I_{j_0})$ contains at most v_0 many roots of $f^{[t]}$ (if there exists no such j_0, then we set $j_0 = s$) . The first part of the chain is $I_0 \supset I_1 \supset \cdots \supset I_{j_0}$ and the second part is $I_{j_0+1} \supset I_{j_0+2} \supset \cdots \supset I_s$. So it is enough to show that $j_0 = O(v_0 \cdot (\log n + \log(\tau + L)))$ and $s - j_0 = O(v_0 \cdot (\log n + \log(\tau + L))$. By using the definition of $s_{\max}$, after every $s_{\max}$ intervals in the chain, the number of roots in the one circle region drops by at least one. I_s has at least one root of $f^{[t]}$ in its one circle region, namely α. Thus $s - j_0 \leq v_0 \cdot s_{\max} = O(v_0 \cdot (\log n + \log(\tau + L)))$. To show a similar upper bound on j_0, consider the following two cases.

Case 1. The first case is when I_{j_0} and I_0 share an endpoint. Thus guarantees the success of the BOUNDARYTEST*. In this case, it can be shown that $j_0 = O(\log n + \log(\tau + L))$. This follows from the proof of Lemma 23 of [SM16], where j_0 appears as s_1 in the proof of Lemma 23 of [SM16].

Case 2. Now consider the cases when I_{j_0} and I_0 do not share share any endpoint. Let $k_0 < j_0$ be the index of the last interval I_{k_0} such that I_{k_0} shares an endpoint with I_0. Then $k_0 = O(\log n + \log(\tau + L))$. Suppose $I_0 = (a_0, b_0)$. We know that all the points in I_{k_0+1} are of absolute value at least $\frac{w(I_{k_0+1})}{4}$. Now it can be seen that after $O(\log n)$ further intervals from k_0, once circle regions of subsequent intervals are contained in the Obreshkoff lens L_n of I_0 as intervals are at least halved in each step. By using Theorem 5.4, we know that the number of roots in L_n is bounded by v_0. Therefore the once circle region of these subsequent intervals contains at most v_0 roots.

This proof is essentially the same as the proof of Lemma 6 and Theorem 7 of [Sag14]. We have only used the fact that the multi points we created in NEWREFINE* are always contained in the intervals spanned by multi-points created in NEWREFINE in [Sag14], as we set $\epsilon = \Theta\left(\frac{1}{n^2}\right)$. Other claims follow directly from the description of NEWREFINE* in Algorithm 5.11. □

Combining Theorem 5.18 and Theorem 5.15, we obtain a bound on the complexity of refining I_0 to a size less than 2^{-L}:

Corollary 5.8. *For refining $I_0 = (a_0, b_0) \subset \mathbb{R}_+$, to a size less than 2^{-L}, the algorithm* NEWREFINE* *needs*

$$\tilde{O}\left(k^5 \cdot (k + \log n) \cdot \log n \cdot \left(k \log n + \tau + L + n\overline{\log}\,(b_0)\right)\right)$$

bit operations. For each endpoint p of the interval returned by NewRefine**, it holds that*

$$\left|M_{\mathcal{D}_f}(p)\right| = 2^{-O(k(k\log n+\tau+L+k))}.$$

5.12 Computing a Weak Covering

We now describe how to compute a weak $(L,[0,1+\frac{1}{n}])$-covering for a given (n,k,τ)-nomial f in polynomial time. Suppose $f=\sum_{i=1}^{k} f_i x^{e_i}$. We first consider an upper bound $\tilde{\tau}\in\mathbb{N}$ for τ with $\tau\le\tilde{\tau}\le\tau+2$, and define $\delta := \min(2^{-2\tilde{\tau}-2},\frac{1}{n})\cdot k^{-2}$. Then, in the first step, we compute $(M_{\mathcal{D}_f}, m[k^2;\delta])-$admissible points a^* and b^* for $m := 2^{-2\tau-2}$ and $m := 1+\frac{2}{n}$, respectively. Then, we follow the approach as outlined in Section 5.9 to compute a weak $(L,[a^*,b^*])$-covering for f, where we use the algorithm NewRefine* from the previous Section to refine isolating intervals for the roots of the fractional derivatives of f to a size less than 2^{-L}. The so obtained covering is indeed also a weak $(L,[0,1+\frac{1}{n}])$-covering for f, which follows from the fact that $b^*\ge 1+\frac{1}{n}$ and each positive root of f is lower bounded by $\left(1+\max_{i=1}^{k-1}\frac{|f_i|}{|f_1|}\right)^{-1} \ge (1+2^{2\tau})^{-1}\ge a^*$, due to Cauchy's root bound (Theorem 5.2). For details, consider the exact description of Algorithm 5.12.

Correctness of the algorithm follows directly from our considerations in Section 5.9. Further notice that, for each i in the outermost for-loop of the algorithm, we add at most $k-i-1$ intervals to W_i to obtain W_{i+1} as $f^{[i]}$ has at most $k-i-1$ positive real roots. Hence, each list W_i contains at most k^2 many intervals. It remains to bound the running time of Algorithm 5.12. The proof of the following Lemma 5.21 follows in a straight forward manner from Theorem 5.15, Corollary 5.8, and the fact that we need to call the refinement algorithm at most k times for each fractional derivative.

Lemma 5.21. *Algorithm 5.12 computes a weak $(L,[0,1+\frac{1}{n}])$-covering for f consisting of at most k^2 many intervals. Its bit complexity is $\tilde{O}(k^7\cdot(k+\log n)\cdot\log n\cdot(k\log n+\tau+L))$.*

Proof. We first prove the claimed bound on the running time. The outermost for loop runs at most $k-1$ times. So we just need to prove the running time bound of $\tilde{O}(k^6\cdot(k+\log n)\cdot(k\log n+\tau+L)\cdot\log n)$ on one iteration of this loop. Note that an interval is added to a weak $(L,[0,1+\frac{1}{n}])$-covering W_i of $f^{[i]}$ if it contains a root of $f^{[i]}$. Also, intervals of W_{i-1} are added to W_i. Thus each interval in W_i corresponds to some positive real root of $f^{[i]}$ or some $f^{[j]}$ for some $j>i$. Each fractional derivative $f^{[i]}$ of any (n,k,τ)-nomial f has exactly $k-i$ monomials. Therefore $f^{[i]}$ has at most $k-i-1$ positive real roots. Hence the total number of intervals in W_i is at most $1+2+\ldots+k+i-1=\binom{k-i}{2}$. Thus, the inner for loop runs for at most $O(k^2)$ iterations.

In each iteration of the inner loop, we compute the sign of $f^{[i]}$ at the end points of some interval (b,c). Note that both b and c are $(M_{\mathcal{D}_f}, m[k^2;\delta'])$-admissible for some

Algorithm 5.12 Compute a weak $(L, [0, 1 + 1/n])$-covering of f

Input: An (n, k, τ)-nomial f and a non-negative integer $L \in \mathbb{N}$.
Output: A weak $(L, [0, 1 + \frac{1}{n}])$-covering of f.

$\delta \leftarrow \frac{1}{k^2} \cdot \min(\frac{1}{n}, 2^{-2\tau-6})$
Compute $(M_{\mathcal{D}_f}, m[k^2; \delta])$-admissible points a^* and b^* for $m \stackrel{\text{def}}{=\!=} 2^{-2\tau-6}$ and $m \stackrel{\text{def}}{=\!=} 1 + \frac{2}{n}$, respectively.
Compute the sign of f at $x = a^*$ and $x = b^*$.
for $i = k - 1$ to 0 **do**
    if $(i = (k - 1))$ **then**
        Compute a trivial weak $(L, [a^*, b^*])$-covering W_{k-1} for $f^{[k-1]}$. ▷ $f^{[k-1]}$ has only one monomial.
        $W_{k-1} \leftarrow \{(a^*, a^*), (b^*, b^*)\}$
    else
        $W_{i+1} \leftarrow$ weak $(L, [a^*, b^*])$-covering for $f^{[i+1]}$ computed in the previous iteration of this loop
        $W_i \leftarrow W_{i+1}$
    end if
    for each consecutive intervals (a, b) and (c, d) in W_{i+1} **do**
        Compute signs of $f^{[i]}(b)$ and $f^{[i]}(c)$.
        if $f^{[i]}(b) f^{[i]}(c) < 0$ **then**
            Use NewRefine* to refine the isolating interval (b, c) to a new interval (b', c') of length at most 2^{-L}.
            $W_i \leftarrow W_i \cup (b', c')$
        end if
    end for
end for
return W_0

$\delta' = 2^{-O(\tau+\log n+L)}$, which follows from Theorem 5.18. Thus by using Lemma 5.17, we know that

$$\left|f^{[i]}(x)\right| \geq 2^{-O(k\cdot(k\log n+\tau+L))}$$

for $x = b, c$. By using Lemma 5.14, we know that we can compute the sign of $f^{[i]}(b)$ and $f^{[i]}(c)$ in time

$$\tilde{O}((k+\log n)\cdot(k(k\log n+\tau+L)+\log n+\tau+k))) = \tilde{O}(k\cdot(k+\log n)\cdot(k\log n+\tau+L)).$$

Here we are using the fact that all these end points c (also b) are in $(0, 1+\frac{3}{n})$ and thus $n\cdot\overline{\log}(c) = O(1)$.

Thus all the signs can be computed in time

$$\tilde{O}(k^2\cdot k\cdot(k+\log n)\cdot(k\log n+\tau+L)).$$

Now note that we perform the refinement on an interval (b, c) using NewRefine* only when (b, c) contains a real root of $f^{[i]}$. Since $f^{[i]}$ has at most $k-i-1$ positive real roots, we use NewRefine* at most $k-i-1$ times. By using Corollary 5.8, all such refinements take time $\tilde{O}(k^6\cdot(k+\log n)\cdot\log n\cdot(k\log n+\tau+L))$. Thus Algorithm Algorithm 5.12 runs in $\tilde{O}(k^7\cdot(k+\log n)\cdot\log n\cdot(k\log n+\tau+L))$ time.

Correctness of W_0 being a weak $(L, [0, 1+1/n])$-covering of f follows from the discussion at the beginning of this section. □

In order to further process a weak $(L, [0, 1+1/n])$-covering for f, we need the intervals in the weak covering to be well separated. For given $L, \lambda \in \mathbb{N}_0$, we say that a list $\mathcal{L}$ of intervals is (L, λ)-*separated* if the distance $\mathrm{dist}(I, J)$ between I and its neighboring intervals is at least $\min(2^{-L}, \lambda\cdot w(I))$. Notice that, starting from an arbitrary list $\mathcal{L}$ of intervals, we can always deduce an (L, λ)-separated list $\mathcal{L}'$ from $\mathcal{L}$ in a way such that each interval in $\mathcal{L}$ is contained in an interval from $\mathcal{L}'$. Namely, this can be achieved by recursively merging pairs of intervals $I, J \in \mathcal{L}$ that violate the above condition until the actual list is (L, λ)-*separated*. See Algorithm 5.13 for details.

By induction, it is easy to see that

$$w(\mathcal{L}') \leq (2+\lambda)^{|\mathcal{L}|}\cdot\max(2^{-L}, w(\mathcal{L})),$$

where $w(\mathcal{L})$ and $w(\mathcal{L}')$ denote the maximal width of an interval in $\mathcal{L}$ and $\mathcal{L}'$, respectively. Hence, by first computing a weak $(L', [0, 1+1/n])$-covering $\mathcal{L}$, with

$$L' = L + k^2\cdot\log(2+\lambda)$$

Algorithm 5.13 Merge

Input: A set of $\mathcal{L}$ intervals of size ℓ and a parameter λ.
Output: A list of intervals $\mathcal{L}'$ such for every interval $I \in \mathcal{L}'$, distance between I and its neighboring intervals is at least $\lambda \cdot w(I)$. Also intervals of $\mathcal{L}'$ cover the intervals of $\mathcal{L}$.

$\mathcal{L}' \leftarrow \mathcal{L}$.
for each interval $I \in \mathcal{L}$ **do**
  if I is not λ-separated **then** ▷ There is some interval J which is "too close" to I
    Suppose J is the neighboring interval of I such that distance between I and J is $< \min(2^{-L}, \lambda \cdot w(I))$.
    $I' \leftarrow$ Smallest interval containing both J and I.
    $\mathcal{L}' \leftarrow \mathcal{L}' \cup I'$.
    $\mathcal{L}' \leftarrow \mathcal{L}' \setminus \{I, J\}$.
    break
  end if
  if Any interval was merged **then**
    return output of this algorithm Algorithm 5.13 on $\mathcal{L}'$.
  end if
  return $\mathcal{L}'$.
end for

and $|L| \leq k^2$, and then recursively merging the intervals, we obtain a weak $(L, [0, 1 + 1/n])-$covering for f that is also (L, λ)-separating and whose intervals have width at most 2^{-L}. From Lemma 5.21, we thus conclude:

Corollary 5.9. *For any $\lambda, L \in \mathbb{N}_0$, we can compute an (L, λ)-separating weak $(L, [0, 1 + 1/n])-$covering for f in*

$$\tilde{O}(k^7(k + \log n) \cdot (k \log n + \tau + L + k^2 \log(2 + \lambda)) \cdot \log n)$$

bit operations.

5.13 T_l-TEST

In the previous section, we have shown how to compute a weak $(L, [0, 1 + \frac{1}{n}))$-covering of a given (n, k, τ)-nomial f. Now, we aim to convert this weak covering to a covering of f. For this, we need an algorithm to count the number of roots of $f(x)$ contained in a given disk. Recent work [Bec+18] introduces a simple algorithm for this task, denoted T_l-test, which is based on Pellet's Theorem (Theorem 5.19).

Theorem 5.19 (Pellet's Theorem, Theorem 28.1 in [Mar66])**.** *Given the polynomial*

$$f(z) = f_0 + f_1 x + \cdots + f_p x^p + \cdots + f_n x^n \quad \text{with} f_p \neq 0.$$

If the polynomial $F_p(x)$ defined by

$$F_p(x) \stackrel{def}{=\!=} |f_0| + |f_1|\, x + \cdots + \left|f_{p-1}\right| x^p - \left|f_p\right| x^p + \left|f_{p+1}\right| x^p + \cdots + |f_n|\, x^n$$

has two positive zeros r and R, $r < R$, then $f(x)$ has exactly p zeros in or on the circle $|x| < r$ and no zeros in the ring $r < |x| < R$.

For an arbitrary polynomial $F \in \mathbb{C}[x]$, a disk $\Delta = \Delta_r(m) \subset \mathbb{C}$, and a parameter $K \geq 1$, we consider the following inequality:

$$T_l(\Delta, K, F) : \left|\frac{F^{(l)}(m)r^l}{l!}\right| - K \cdot \sum_{i \neq l} \left|\frac{F^{(i)}(m)r^i}{i!}\right| > 0. \tag{5.8}$$

Hence, we check whether the absolute value of the l^{th} coefficient $a_l = \frac{F^{(l)}(m)r^l}{l!}$ of

$$F_\Delta(x) = f(m + rx) = \sum_{i=0}^{n} a_i x^i = \sum_{i=0}^{n} \frac{F^{(i)}(m)r^i}{i!} x^i$$

dominates the sum of the absolute values of all remaining coefficients weighted by the parameter K. We say that $T_l(\Delta, K, F)$ succeeds if the above inequality is fulfilled. Otherwise, we say that it fails. In case of success for any $K \geq 1$, Δ contains exactly l roots of F counted with multiplicity. This follows directly from Rouché's Theorem (Theorem 5.9) applied on $a_l x^l$ and $F_\Delta(x)$. We obtain no information in case of a failure of $T_l(\Delta, K, F)$. In [Bec+18], sufficient conditions were derived on the success of the T_l-test:

Theorem 5.20 (Corollary 1 in [Bec+18])**.** *Let $F \in \mathbb{C}[x]$ be a polynomial of degree n, and $\Delta_r(m)$ be a disk. If $\Delta_r(m)$ as well as the enlarged disk $\Delta_{256n^5 r}(m)$ contain l roots of F counted with multiplicity, then $T_l(\Delta_{16nr}(m), \frac{3}{2}, F)$* succeeds.

Unfortunately, the above test has two major drawbacks when dealing with sparse polynomials. First, we need to compute the coefficients F_Δ exactly, which we cannot afford as the bit-size of each coefficient of $F_\Delta(x)$ is $\Omega(n)$. Second, an even more severe, there are n coefficients to be computed. Hence, using the above approach directly to count the number of roots of a sparse polynomial f does not work. Instead, we propose two modifications to overcome these issues. The first modification, namely to use approximate (in a proper manner) instead of exact arithmetic, has already been considered in previous work. However, the second modification is more subtle. It exploits the fact that, for a suitably chosen disk centered at some admissible point, only the first k^2 coefficient are relevant for the outcome of the above test.

We first go into details with respect to our first modification. Let us define $E_\ell \stackrel{\text{def}}{=\!=} |a_l|$ and $E_r \stackrel{\text{def}}{=\!=} K \cdot \sum_{i \neq l} |a_i|$ the expressions on the left and right hand side of the inequality

in (Equation (5.8)). We aim to check whether $E_\ell - E_r > 0$ or not. In general, if a predicate $\mathcal{P}$ is of the latter form $\mathcal{P} = (E_\ell - E_r > 0)$ with two (computable) expressions E_ℓ and E_r, you can compute approximations $\tilde{E}_\ell$ and $\tilde{E}_r$ of E_ℓ and E_r with $\left|\tilde{E}_\ell - E_\ell\right| < 2^{-L}$ and $\left|\tilde{E}_r - E_r\right| < 2^{-L}$ for $L = 1, 2, 4, \ldots$. For a certain L, you may then try to compare E_ℓ and E_r taking into account their corresponding approximations and the approximation error. Eventually (for a sufficiently large L), you either succeed, in which case you can return the sign, or assert that E_ℓ and E_r must be good approximations of each other. In the latter case, you just return a flag called Undecided. In short, this is the idea of so-called *soft-predicates*. For details, we refer to [Bec+18]. Notice that, in cases where

Algorithm 5.14 Soft Predicate $\tilde{\mathcal{P}}$.

Input: A predicate $\mathcal{P}$ defined by non-negative expressions E_ℓ and E_r , with $E_\ell \neq 0$ or $E_r \neq 0$, *i.e.*, $\mathcal{P}$ succeeds if and only if $E_\ell > E_r$. A rational constant $\delta > 0$.

Output: True, False, or Undecided. In case of True (False), $\mathcal{P}$ succeeds (fails). In case of Undecided, we have $\frac{1}{1+\delta} \cdot E_\ell < E_r \leq (1+\delta) \cdot E_\ell$.

$L \leftarrow 1$
while True **do**
  Compute L-bit approximations $\tilde{E}_\ell$ and $\tilde{E}_r$ of E_ℓ and E_r.
  Compute upper and lower bounds of E_ℓ and E_r as below. ▷ E_ℓ^+, E_r^+ are upper bounds and E_ℓ^-, E_r^- are lower bounds.
  $E_\ell^+ \leftarrow \max(0, \tilde{E}_\ell + 2^{-L})$
  $E_r^+ \leftarrow \max(0, \tilde{E}_r + 2^{-L})$
  $E_\ell^- \leftarrow \max(0, \tilde{E}_\ell - 2^{-L})$
  $E_r^- \leftarrow \max(0, \tilde{E}_r - 2^{-L})$
  if $E_\ell^- > E_r^+$ **then** ▷ In this case, we are sure that $E_\ell > E_r$
    return True
  end if
  if $E_\ell^+ < E_r^-$ **then** ▷ In this case, we are sure that $E_\ell < E_r$
    return False
  end if
  if $\left(\frac{1}{1+\delta}\right) \cdot E_\ell^+ \leq E_r^- \leq E_r^+ \leq (1+\delta) \cdot E_\ell^-$ **then** ▷ In this case, we know that E_ℓ, E_r are good approximations of each other.
    return Undecided.
  end if
  $L \leftarrow 2 \cdot L$
end while

E_ℓ considerably differs from E_r, the soft predicate $\tilde{\mathcal{P}}$ allows us to compute the sign of $\mathcal{P}$ without the need of exact arithmetic. In all other cases (if it returns Undecided), we know at least that E_ℓ and E_r are good approximations of each other. We remark that, in [Bec+18], the above soft predicate $\tilde{\mathcal{P}}$ was only described for $\delta = \frac{1}{2}$, however, it easily generalizes to any constant δ. The proof in [Bec+18] directly shows that, for

any constant δ, Algorithm 5.14 needs an L_0-bit approximation of E_ℓ and E_r with L_0 bounded by:

$$L_0 \leq 2 \cdot \left(\overline{\log} \left(\max(E_\ell, E_r)^{-1} \right) + \overline{\log} \left(\frac{1}{\delta} \right) \right).$$

In [Bec+18], a soft-variant of the T_l-test was considered, where the authors compared the expressions $E_\ell \overset{\text{def}}{=\!=} |a_l|$ and $E_r \overset{\text{def}}{=\!=} \sum_{i \neq l} |a_i|$. Now, we apply the above soft-predicate to the expressions $E_\ell \overset{\text{def}}{=\!=} a_l$ and $E_r \overset{\text{def}}{=\!=} \sum_{i \neq l}^{i \leq k^2} |a_i|$, that is, we replace the entire sum $\sum_{i \neq l} |a_i|$ by its truncation after the first k^2 terms. However, we will make the assumption that the truncated sum $\sum_{i > k^2} |a_i|$ is upper bounded by $\frac{|a_0|}{128}$; see Algorithm 5.15. This might look haphazardly at first sight, however, we will later see that the latter condition is always fulfilled for a k-nomial F and a suitable disk $\Delta_r(m)$ centered at an admissible point.

Algorithm 5.15 $\tilde{T}_l$-test

Input: An (n, k, τ)-nomial $f(x)$, a disk $\Delta := \Delta_r(m)$ in the complex space and an integer l with $0 \leq l \leq k$. It is required that $\sum_{i > k^2} |a_i| \leq \frac{|a_0|}{128}$, where $f_\Delta(x) = \sum_{i=0}^{n} a_i \cdot x^i$.

Output: True, False or Undecided. If the algorithm returns True then the disk $\Delta_r(m)$ contains exactly l roots.

1: Define $E_\ell \overset{\text{def}}{=\!=} |a_l|$ and $E_r \overset{\text{def}}{=\!=} \frac{65}{64} \cdot \sum_{i \neq l}^{i \leq k^2} |a_i|$.
2: Define predicate $\mathcal{P} = (E_\ell - E_r > 0)$.
3: **return** output of Algorithm 5.14 on predicate $\mathcal{P}$ with $\delta = \frac{1}{128}$.

Lemma 5.22. *For a disk $\Delta := \Delta_r(m) \subset \mathbb{C}$, the $\tilde{T}_l$-test (Algorithm 5.15) needs to compute L-bit approximations of E_ℓ and E_r with*

$$L \leq L(m, r, f) \overset{def}{=\!=} 2 \cdot \left(\overline{\log} \left(\max(E_\ell, E_r)^{-1} \right) + 8) \right).$$

If $T_l(\Delta, \frac{3}{2}, f)$ succeeds and $\sum_{i > k^2} |a_i| \leq \frac{|a_0|}{128}$, then the $\tilde{T}_l$-test returns True. Running Algorithm 5.15 for all $l = 0, \ldots, k$ uses a number of bit operations upper bounded by:

$$\tilde{O} \left(k^2 \cdot (k + \log n)(L(m, r, f) + \tau + n\overline{\log}(m) + k^2 \cdot (\log n + \overline{\log}(r))) \right).$$

Proof. We first prove correctness. If the algorithm returns True, then $E_\ell > E_r$, and thus $|a_l| > \frac{65}{64} \cdot \sum_{i \neq l}^{i \leq k^2} |a_i|$. If $l = 0$, then

$$\sum_{i \neq 0} |a_l| < \frac{64}{65} \cdot |a_0| + \frac{1}{128} \cdot |a_0| < |a_0|.$$

Otherwise, we have

$$|a_l| > \frac{65}{64} \cdot \sum_{i \neq l}^{i \leq k^2} |a_i| \geq \sum_{i \neq l}^{i \leq k^2} |a_i| + \frac{1}{64} \cdot |a_0| \geq \sum_{i \neq l}^{i \leq n} |a_i| .$$

Hence, in both cases, $T_l(\Delta, 1, f)$ succeeds, which implies that Δ contains exactly l roots.

Now, suppose that $T_l(\Delta, \frac{3}{2}, f)$ succeeds. If the $\tilde{T}_l$-test returns Undecided, then $\frac{128}{129} \cdot E_\ell < E_r \leq \frac{129}{128} \cdot E_\ell$. On the other hand, we have

$$|a_l| > \frac{3}{2} \sum_{i \neq l}^{\leq n} |a_i| \geq \frac{3}{2} \sum_{i \neq l}^{\leq k^2} |a_i|$$

and thus $E_\ell > \frac{3}{2} E_r$, which contradicts the fact that $\frac{128}{129} \cdot E_\ell < E_r$. If the $\tilde{T}_l$-test returns False, a similar argument yields a contradiction as well. This shows that success of T_l implies that $\tilde{T}_l$ returns True. It remains to show the claimed bounds on the bit complexity. It suffices to estimate the cost for computing an $L(m, r, f)$-bit approximations of E_ℓ and E_r. The i^{th} coefficient a_i, with $i \leq k^2$, can be computed by evaluating the $(n, k, \tau + k^2 \cdot (\log n + \overline{\log}(r)))$-nomial $g_i = \frac{f^{(i)}(x) r^i}{i!}$ at $x = m$. In order to compute $L(m, r, f)$-bit approximations of E_ℓ and E_r, we need to compute an $(L(m, r, f) + 2 \log k)$-bit approximation of each $g_i(m)$, for $i = 0, \ldots, k$. According to Lemma 5.14, this can be done using

$$\tilde{O}(k^2 \cdot (k + \log n)(L(m, r, f) + n\overline{\log}(m) + \tau + k^2 \cdot (\log n + \overline{\log}(r)))$$

bit operations. □

Notice that, in order to actually use the $\tilde{T}_l$-test for counting the roots in a disk Δ, we need the following to be satisfied: $\sum_{i > k^2} |a_i| \leq \frac{|a_0|}{128}$ to be true.

Theorem 5.21. *Let F be a (n, k, τ)-nomial with $n \geq k \geq 2$. Let $\Delta \stackrel{def}{=\!=} \Delta_r(m)$ be a disk centered at some $m \in \mathbb{R}^+$ with $\frac{m}{r} > n^{16}$, and let $F_\Delta(x) = \sum_{i=0}^{n} a_i \cdot x^i$. Further suppose that $\Delta_{\frac{r}{2 \cdot k^{4k+2}}}(m)$ does not contain any root of F and $F(m) \neq 0$. Then, it holds that $\sum_{i > k^2} |a_i| \leq \frac{|a_0|}{128}$.*

Proof. Let $z_1, z_2, \ldots, z_n$ be the complex roots of $F(x)$, then we have:

$$\frac{a_i}{a_0} = \frac{F^{(i)}(m)}{F(m) \cdot i!} \cdot r^i = \frac{r^i}{i!} \cdot \sum_{(j_1, j_2, \ldots, j_i)} \frac{1}{\prod_{\ell=1}^{i} (m - z_{j_\ell})}.$$

Here we sum over all tuples $(j_1, j_2, \ldots, j_i)$ with distinct entries j_s, $1 \leq j_s \leq n$. For a fixed tuple $(j_1, j_2, \ldots, j_i)$, at most k of the i roots $z_{j_1}, z_{j_2}, \ldots, z_{j_i}$ are contained in the cone C_n as defined in Figure 5.2, whereas the remaining $i - k$ roots are located outside of C_n. Since $\frac{m}{r} > n^{16}$, the distance from m to any of these roots is at least $n^{15} r$. Also, since $\Delta_{\frac{r}{2 \cdot k^{4k+2}}}(m)$

does not contain any roots of $F(x)$, distance of m from the roots in C_n is at least $\frac{r}{2\cdot k^{4k+2}}$. Thus, we get $\sum_{(j_1,j_2,\ldots,j_i)} \frac{1}{\prod_{\ell=1}^{i} |m-z_{j_k}|} \leq \binom{n}{i} \cdot \frac{2^k \cdot k^{4k^2+2k}}{r^k \cdot (n^{15}r)^{i-k}}$. Hence, for $i > k^2$, we get

$$\begin{aligned}
\frac{|a_i|}{|a_0|} &\leq \frac{r^i}{i!} \cdot \binom{n}{i} \cdot \frac{2^k \cdot k^{4k^2+2k}}{r^k \cdot (n^{15}r)^{i-k}} = \frac{1}{i!} \cdot \binom{n}{i} \cdot \frac{2^k \cdot k^{4k^2+2k}}{n^{15(i-k)}} \\
&\leq \frac{1}{i! \cdot i!} \cdot \frac{2^k \cdot k^{4k^2+2k}}{n^{14i-15k}} \\
&\leq \frac{1}{i! \cdot i!} \cdot \frac{2^k \cdot k^{4k^2+2k}}{n^{6i}} \qquad \text{(By using the fact that } \tbinom{n}{i} \leq \tfrac{n^i}{i!} \text{ and } 15k < 8k^2 < 8i\text{)} \\
&\leq \frac{1}{i! \cdot n^6 \cdot i!} \cdot \frac{2^k \cdot k^{4k^2+2k}}{k^{6k^2}} \leq \frac{2^k}{n^6 \cdot (i!)^2} \cdot \left(\frac{1}{k}\right)^{1-2k^2+2k} < \frac{1}{128n}
\end{aligned}$$

Hence, summing up over all $i > k^2$ proves the claim. □

The following Corollary is now an immediate consequence of Theorem 5.21 and Lemma 5.15.

Corollary 5.10. *Let $f(x) \in \mathbb{R}[x]$ be an (n,k,τ)-nomial, and $m, r \in \mathbb{R}^+$. Further assume that $\frac{m}{r} \geq 2(1+n^{16})$. Let m^* be a $(f, m[k^2; \frac{r}{k^2}])$ -admissible point and $r^* = 2r$. Define $\Delta = \Delta_{r^*}(m^*) \supseteq \Delta_r(m)$ and $f_\Delta(x) = \sum_{i=0}^{n} a_i \cdot x^i$. , then $\sum_{i>k^2} |a_i| \leq \frac{|a_0|}{128}$.*

In the next step, we show how to satisfy the precondition of the T_l-test. Theorem 5.20 says that if $\Delta_{256n^5 r}(m)$ does not contain any of the roots which are not contained in $\Delta_r(m)$, then $T_l(\Delta_{16nr}, \frac{3}{2}, f)$ succeeds for some l. Let us define $M \stackrel{\text{def}}{=\!=} 256n^5$, and let $\Delta_i \stackrel{\text{def}}{=\!=} \Delta_{M^i r}(m)$ for $i = 0, 1, \ldots, k+1$. Further assume that r has been chosen sufficiently small enough such that each of the disks Δ_i is contained in the cone C_n. Since C_n contains at most k roots, there must exist a j with $0 \leq j \leq k$ such that $\Delta_{j+1} - \Delta_j$ does not contain any root. Hence the T_l-test will succeed on $\Delta_{16nM^j r}(m)$. So instead of running the T_l-test on some initial disk $\Delta_r(m)$, we run it on all disks $\Delta_{16nM^i r}(m)$ for $i = 0, 1, \ldots, k$. See Figure 5.3.

We return the first disk on which the T_l-test succeeds; see Algorithm 5.16.

Correctness of the algorithm follows immediately from the above discussion. The condition on m and r guarantees that each of the disks Δ_i is contained in C_n. Lemma 5.23 gives a bound on its running time.

Lemma 5.23. *Algorithm 5.16 returns a disk $\Delta_{r'}(m')$, with $r' \leq \frac{2Rr}{n^{15}}$ and $m' \in [m - \frac{r'}{32n}, m + \frac{r'}{32n}]$, together with the number of roots of $f(x)$ in $\Delta_{r'}(m')$. Its bit complexity is bounded by*

$$\tilde{O}\left(k^5 \cdot (k + \log n) \cdot \left(k^2 \log n + n\overline{\log}\left(m + r2^{8k+4}n^{5k+16}\right) + \tau + \overline{\log}\left(r^{-1}\right)\right)\right).$$

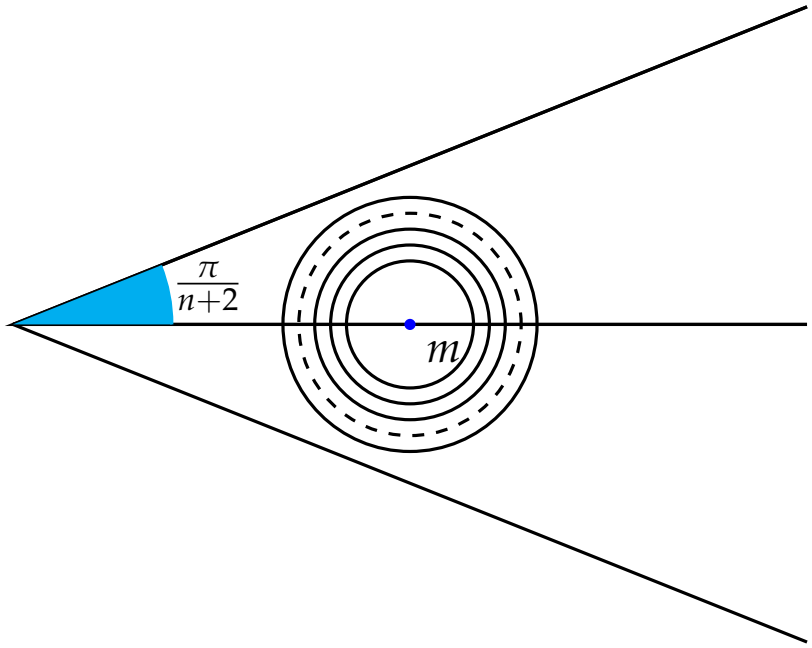

Figure 5.3: Here $M = 256n^5r$ and innermost circle has radius M and circles after that have radius Mr, $M^2r, \ldots$

Algorithm 5.16 Wrapper $\tilde{T}_l$-test

Input: An (n, k, τ)-nomial $f(x)$, a disk $\Delta := \Delta_r(m)$ in the complex space. We assume $m \geq 2r + 2Rnr$ with $R = 2^{8k+4}n^{5k+16}$.

Output: A disk $\Delta_{r'}(m') \supseteq \Delta_r(m)$ and the number of the roots of f in $\Delta_{r'}(m')$, where m' and r' satisfy: $r' \leq \frac{2Rr}{n^{15}}$, $m' \in [m - \frac{r'}{32n}, m + \frac{r'}{32n}]$.

1: $M \leftarrow 256n^5r'$
2: **for** each $0 \leq i \leq k$ **do**
3: **for** each $0 \leq l \leq k$ **do**
4: $r_i \leftarrow M^i r$
5: Compute an $(f, m[k^2; \frac{r_i}{k^2}])$-admissible point m_i^*.
6: $r_i^* \leftarrow 2r_i$
7: Perform the $\tilde{T}_l$-test, that is Algorithm 5.15, on $\Delta_{16nr_i^*}(m_i^*)$.
8: **if** $\tilde{T}_l$-test succeeded in the previous step **then**
9: **return** $\Delta_{16nr_i^*}(m_i^*)$ and l.
10: **end if**
11: **end for**
12: **end for**

Proof. The condition $m \geq 2r + 2Rnr$ implies that all the disks considered in the Algorithm 5.16 are contained in the cone C_n. In addition, the condition of Corollary 5.10 is also fulfilled.

We have that $M = 256n^5$, and $\Delta_i \overset{\text{def}}{=\!=} \Delta_{M^i r}(m)$ for $i = 0, 1, \ldots, k+1$. Since C_n contains at most $k-1$ roots (Theorem 5.5) of f, we know that there exists an i such that Δ_i and Δ_{i+1} both contain l roots of f. Thus line 7 would succeed for this i. It is clear that we make at most k^2 calls to line 7. By using Lemma 5.22, line 7 runs in

$$\tilde{O}\left(k^2 \cdot (k + \log n)(L + \tau + n\overline{\log}\,(m_i^*) + k^2 \cdot (\log n + \overline{\log}\,(16nr_i^*)))\right)$$

bit operations. Here $L \overset{\text{def}}{=\!=} 2 \cdot \left(\overline{\log}\,(\max(E_\ell, E_r)^{-1}) + 8\right)$ with $a_i = \frac{f^{(i)}(m')}{i!} \cdot (16nr_i^*)^i$ and E_ℓ, E_r defined as in Algorithm 5.15. Note that $\max(E_\ell, E_r) \geq a_0$. By using Theorem 5.17, we get that $a_0 = f(m') \geq 2^{-O\left(k \cdot \left(k \log n + \tau + \overline{\log}\left(\frac{k^2}{r_i}\right)\right)\right)}$ where r_i is as defined in line 4. Therefore

$$\max(E_\ell, E_r) \geq 2^{-O\left(k \cdot \left(k \log n + \tau + \overline{\log}\left(\frac{k^2}{r_1}\right)\right)\right)}.$$

Thus the total running time of line 7 is

$$\tilde{O}\left(k^3 \cdot (k + \log n) \cdot \left(k^2 \log n + n\overline{\log}\,(m + Rr) + \tau + \overline{\log}\left(r^{-1}\right)\right)\right).$$

Now the claimed running time follows. □

5.14 Computing a Covering

We now show how to compute an $(L, [0, 1 + \frac{1}{n}])$-covering from a weak $(L', [0, 1 + \frac{1}{n}])$-covering. For this, we apply Algorithm 5.16 to the one-circle regions of these intervals in the weak covering. The following Lemma 5.24 shows that the requirements in Algorithm 5.16 are fulfilled if we choose L' large enough. In addition, by ensuring that the intervals in the weak covering are well separated from each other, we can ensure that the corresponding disks returned by Algorithm 5.16 are disjoint.

Lemma 5.24. *Algorithm 5.17 computes an $(L, [0, 1 + \frac{1}{n}])$-covering $\mathcal{L}'$ for f using*

$$\tilde{O}\left(k^7 \cdot (k + \log n)(k^3 \log n + \tau + L)\right)$$

bit operations. The distance between any two disks of $\mathcal{L}'$ is at least $32 \cdot 2^{-L}$, and $\Delta \cap \mathbb{R} \subset (2^{-3\tau}, 2)$ for any disk Δ in $\mathcal{L}'$.

Proof. By using Corollary 5.9, line 3 in Algorithm 5.17 can be performed using

$$\tilde{O}(k^7(k + \log n) \cdot (k \log n + \tau + L + \lceil \log R \rceil + 4\tau + 5 + \log n + k^2 \log(2 + 8R)) \cdot \log n)$$

Algorithm 5.17 Computing an $(L, [0, 1 + \frac{1}{n}])$-covering

Input: An (n, k, τ)-nomial $f(x)$ and a positive integer L.
Output: An $(L, [0, 1 + 1/n])$-covering for f.

1: $R \leftarrow 2^{8k+4} n^{5k+16}$
2: $L' \leftarrow L + \lceil \log R \rceil + 4\tau + 5 + \log n$
3: Compute a weak $(L', [0, 1 + \frac{1}{n}])$-covering $\mathcal{L}$ for f that is $(L', 8R)$-separated. ▷ By using Corollary 5.9.
4: $\mathcal{L}' \leftarrow \emptyset$
5: **for** each interval $I = (a, b) \in \mathcal{L}$ **do**
6: $\quad \Delta \leftarrow \Delta_{\frac{b-a}{2}}(\frac{a+b}{2})$=one circle region of I
7: $\quad (\Delta_{r'}(m'), \mu) \leftarrow$ Output of Algorithm 5.16 on f and Δ.
8: $\quad \mathcal{L}' \leftarrow \mathcal{L}' \cup \{(\Delta_{r'}(m'), \mu)\}$
9: **end for**

bit operations. By using R=$2^{8k+4} n^{5k+16}$, this bound writes as

$$\tilde{O}\left(k^7 \cdot (k + \log n)(k^3 \log n + \tau + L)\right).$$

Note that the weak covering returned in line 3 is of size at most k^2. By using Lemma 5.23, we need

$$\tilde{O}\left(k^7 \cdot (k + \log n) \cdot \left(k^2 \log n + n\overline{\log}\,(m + Rr) + \tau + \overline{\log}\left(r^{-1}\right)\right)\right)$$

many bit operations to convert the weak covering to a strong covering, where r is the minimum size of the intervals in the weak covering we computed in line 3. We know that $r = 2^{-O(L')}$.Therefore $m + Rr \leq 1 + \frac{1}{n^{O(1)}}$. Thus $n\overline{\log}\,(m + Rr) = O(1)$. Hence the total running time is $\tilde{O}\left(k^7 \cdot (k + \log n)(k^3 \log n + \tau + L)\right)$. □

It remains to show how to compute an $(L, [0, \infty))$-covering for f from an $(L, [0, 1 + \frac{1}{n}))$-covering $\mathcal{L}_1$ for f and an $(L, [0, 1 + \frac{1}{n}))$-covering $\mathcal{L}_2$ for $x^n f(\frac{1}{x})$. We first derive an $(L, [\frac{n}{n+1}, \infty))$-covering for f from $\mathcal{L}_2$ by inverting the disks Δ in $\mathcal{L}_2$. The proof of the following Lemma 5.25 is straight forward.

Lemma 5.25. *Let $\mathcal{L}$ be an $(L, [0, 1 + \frac{1}{n}])$-covering of $x^n f(\frac{1}{x})$ as computed by Algorithm 5.17, and $\mathcal{L}' \stackrel{def}{=\!=} \{(\Delta^{-1}, \mu) : (\Delta, \mu) \in \mathcal{L}\}$ be the list obtained from $\mathcal{L}$ by inverting the disks in $\mathcal{L}$ (i.e. $\Delta_r(m)^{-1} = \Delta_{r'}(m')$ with $r' = \frac{r}{m^2 - r^2}$ and $m' = \frac{m}{m^2 - r^2}$). Then, $\mathcal{L}'$ is an $(L', [\frac{n}{n+1}, \infty))$-covering of f with $L' \geq L - 6\tau$, and the distance between two disks in $\mathcal{L}'$ is at least $8 \cdot 2^{-L}$.*

Proof. It is easy to see that $\Delta_{r'}(m')$ is the image of the disk $\Delta_r(m)$ under the map $z \mapsto \frac{1}{z}$. Since the value of the left end point $m - r$ of any disk in $\mathcal{L}$ is at least $2^{-3\tau}$, we obtain that:

$$m^2 - r^2 = (m-r)(m+r) \geq (m-r)^2 \geq 2^{-6\tau}.$$

Therefore:

$$r' = \frac{r}{m^2 - r^2} \leq r2^{6\tau}.$$

Hence $L' \geq L - 6\tau$.

Suppose t is a real root of $f(x)$ with $t \in [\frac{n}{n+1}, \infty)$. Then $\frac{1}{t}$ is the root of $x^n f(\frac{1}{x})$ and also $\frac{1}{t} \in [0, 1 + \frac{1}{n}])$. Therefore $\frac{1}{t}$ is covered by $\mathcal{L}$. Hence $\mathcal{L}'$ is $(L', [\frac{n}{n+1}, \infty))$-covering of f.

Suppose $\Delta_{r_1}(m_1), \Delta_{r_2}(m_2)$ are two disks in $\mathcal{L}$. The distance between the corresponding inverted disks $\Delta_{r_1'}(m_1'), \Delta_{r_2'}(m_2')$ is:

$$\frac{1}{m_1 + r_1} - \frac{1}{m_2 - r_2} \geq \frac{(m_2 - r_2) - (m_1 + r_1)}{(m_2 - r_2) \cdot (m_1 + r_2)}$$

Since the value of the right end point of every disk in $\mathcal{L}$ is at most 2, it follows that $(m_2 - r_2) \cdot (m_1 + r_2) \leq 4$. Therefore:

$$\frac{1}{m_1 + r_1} - \frac{1}{m_2 - r_2} \geq \frac{(m_2 - r_2) - (m_1 + r_1)}{(m_2 - r_2) \cdot (m_1 + r_2)} \geq \frac{32 \cdot 2^{-L}}{4} = 8 \cdot 2^{-L}.$$

Above, we have used the fact that the distance $(m_2 - r_2) - (m_1 + r_1)$ between any two $\Delta_{r_1}(m_1), \Delta_{r_2}(m_2)$ disks in $\mathcal{L}$ is at least $32 \cdot 2^{-L}$ (by using Lemma 5.24). □

Finally, we merge an $(L, [0, 1 + \frac{1}{n}))$-covering $\mathcal{L}_1$ and an $(L, [\frac{n}{n+1}, \infty))$-covering $\mathcal{L}_2$ for f. Here, we assume that $L > 3 + \log n$, and that the coverings are computed using Algorithm 5.17 and by inverting the $(L, (0, 1 + \frac{1}{n}))$-covering for $x^n \cdot f(\frac{1}{x})$ to obtain $\mathcal{L}_2$. This guarantees that the distance between any two disks in either $\mathcal{L}_1$ or $\mathcal{L}_2$ is at least $8 \cdot 2^{-L}$. For the merge, we keep each disk from $\mathcal{L}_1$ that has no intersection with a disk from $\mathcal{L}_1$, and vice versa. For each pair of elements $(\Delta_1, \mu_1) \in \mathcal{L}_1$ and $(\Delta_2, \mu_2) \in \mathcal{L}_2$ with $\Delta_1 \cap \Delta_2 \neq \emptyset$, we keep (Δ_1, μ_1) (and omit (Δ_2, μ_2)) if the center of Δ_1 is not larger than 1. Otherwise, we keep (Δ_2, μ_2) (and omit (Δ_1, μ_1)). Following this approach, we might loose some of the complex roots that are contained in the union of Δ_1 and Δ_2, however, we will not loose any real root. Now we show that we do not loose any real root.

Suppose $\Delta_1 \cap \Delta_2 \neq \emptyset$ and we kept (Δ_1, μ_1). Assume that there is a real root $a \in \Delta_2$ which we lost by following this approach. We have $\Delta_1 = \Delta_r(m)$. Since we kept (Δ_1, μ_1), we know that $m < 1$. Since $L > 3 + \log n$, radius of both Δ_1, Δ_2 is at most $\frac{1}{8n}$. It follows that $|m - a| < \frac{3}{8n}$. In particular, $a < 1 + \frac{1}{n}$. This means there exists a pair $(\Delta, \mu) \in \mathcal{L}_1$(because $\mathcal{L}_1$ is an $(L, [0, 1 + \frac{1}{n}))$-covering) such that $a \in \Delta$. In this case, the distance between Δ_1 and Δ is less than $8 \cdot 2^{-L}$. This is a contradiction. A similar argument applies when we keep Δ_2 instead of Δ_1.

Thus, the so obtained list constitutes an $(L, (0, \infty))$-covering for f.

Notice that any two $(L,(0,\infty))$ and $(L,(-\infty,0))$-coverings for f can be trivially merged by taking their union as there are no intersections. In addition, since the final covering contains a list of disjoint disks contained in the union of the cone C_n and its reflection on the imaginary axis, and since the union of these two cones contains at most $2k-1$ roots of f, the number of disks is also bounded by $2k-1$. Hence, our main Theorem Theorem 5.13 follows.

Part III

COMPLEXITY OF SYMMETRIC POLYNOMIALS

6

COMPLEXITY OF SYMMETRIC POLYNOMIALS

This chapter deals with arithmetic complexity of symmetric polynomials. It is known that symmetric Boolean functions are "easy" to compute. So one can ask the same question for symmetric polynomials. In this chapter, we show that there exist symmetric polynomials which are "hard" to compute, that is, there exist symmetric polynomial families having "large" arithmetic circuit complexity (assuming VP $\neq$ VNP). For this purpose, we use an algebraic version of Newton method used in finding the roots of uni-variate polynomials.

6.1 Checking Symmetries

First we study the complexity of checking whether a given Boolean function or polynomial is symmetric? To this end we define the following problem: SFT (symmetric function testing).

Problem 6.1 (SFT). *Given a Boolean circuit C computing the Boolean function $f(x_1, x_2, \ldots, x_n)$, check if f is symmetric, that is, is $f(x_1, x_2, \ldots, x_n) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$ for all $\sigma \in \mathfrak{S}_n$?*

We now show that SFT is a "hard problem".

Lemma 6.1. SFT *and* CSAT *are polynomial time Turing reducible to each other, i.e.,* SFT $\leq_{\mathrm{P}}^{\mathrm{T}}$ CSAT *and* CSAT $\leq_{\mathrm{P}}^{\mathrm{T}}$ SFT.

Proof. Given a Boolean circuit C, we want to check if the function $f(x_1, x_2, \ldots, x_n)$ computed by C is symmetric. As the permutation group $\mathfrak{S}_n$ is generated by two permutations $\sigma \stackrel{\text{def}}{=\!=} (1,2)$ and $\pi \stackrel{\text{def}}{=\!=} (1,2,\ldots,n)$ [Bra+11], it is necessary and sufficient to check if the given function f is invariant under these two permutations of variables. Thus we define the following Boolean functions:

$$g(x_1, x_2, \ldots, x_n) \stackrel{\text{def}}{=\!=} f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$$
$$h(x_1, x_2, \ldots, x_n) \stackrel{\text{def}}{=\!=} f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$$

Now note that the equality of two Boolean variables x, y can be checked by the following equality gadget.

$$(x \stackrel{?}{=} y) = (\neg x \vee y) \wedge (x \vee \neg y)$$

Thus we only need to check if both $(\neg f \vee g) \wedge (f \vee \neg g)$ and $(\neg f \vee h) \wedge (f \vee \neg h)$ are tautologies (always equal to 1). This can be checked by two oracles calls to CSAT. Thus SFT $\leq_{\mathrm{P}}^{\mathrm{T}}$ CSAT.

Now we prove the other direction. Given a Boolean circuit C, we want to check if the function $f(x_1, x_2, \ldots, x_n)$ computed by C is always zero. First we make an oracle call to SFT to check if f is symmetric. If f is not symmetric then obviously f is a non-zero function because the zero function is trivially symmetric. Thus we can assume f to be symmetric. Now we ask the SFT oracle if the function $h \stackrel{\text{def}}{=\!=} f \wedge x_1$ is symmetric? If f was the zero function then so is h, therefore SFT oracle will answer that h is symmetric. So if SFT oracle answers h to be non-symmetric then obviously f was non-zero. If h also turns out to be symmetric then we know that:

$$\forall (a_1, a_2, \ldots, a_n) \in \{0,1\}^n : f \wedge a_1 = f \wedge a_2 = \cdots = f \wedge a_n. \tag{6.1}$$

Suppose f evaluated to 1 on a point $(a_1, a_2, \ldots, a_n) \notin \{(0,0,\ldots,0),(1,1,\ldots,1)\}$. This means that there exists $(a_1, a_2, \ldots, a_n) \in \{0,1\}^n$ such that $f(a_1, a_2, \ldots, a_n) = 1$ with $a_i = 1, a_j = 0$ for some $i, j \in [n]$. Then obviously we have $f(a_1, a_2, \ldots, a_n) \wedge a_i = 1$ and $f(a_1, a_2, \ldots, a_n) \wedge a_j = 0$. Hence Equation (6.1) can not to be true. Thus f can only be non-zero on the set $\{(0,0,\ldots,0),(1,1,\ldots,1)\}$. The value of f at both these points can be checked manually to check whether f is the zero function or not. Therefore CSAT $\leq_{\mathrm{P}}^{\mathrm{T}}$ SFT. □

Analogous to SFT, we define the following problem: SPT (symmetric polynomial testing).

Problem 6.2 (SPT). *Given an arithmetic circuit C computing the polynomial $f(x_1, x_2, \ldots, x_n)$, check if f is a symmetric polynomial?.*

Lemma 6.2. SPT *and* ACIT *are polynomial time many one reducible to each other, i.e.,* SPT $\leq_{\mathrm{P}}$ ACIT *and* ACIT $\leq_{\mathrm{P}}$ SPT.

Proof. Given an arithmetic circuit C, we want to check if the polynomial $f(x_1, x_2, \ldots, x_n)$ computed by C is symmetric.

As in the proof of the Lemma 6.1, we use the fact that permutation group $\mathfrak{S}_n$ is generated by two permutations $\sigma \stackrel{\text{def}}{=\!=} (1,2)$ and $\pi \stackrel{\text{def}}{=\!=} (1,2,\ldots,n)$, it is necessary and sufficient to check if the given polynomial f is invariant under these two permuta-

tions of variables. Analogous to the proof of the Lemma 6.1, we define the following polynomials:

$$g(x_1, x_2, \ldots, x_n) \stackrel{\text{def}}{=\!=} f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$$
$$h(x_1, x_2, \ldots, x_n) \stackrel{\text{def}}{=\!=} f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$$

Thus f is symmetric iff $f - g = f - h = 0$. Consider the polynomial $F = y(f - g) + z(f - h)$, where y, z are fresh variables. Thus f is symmetric iff F is the zero polynomial. Hence SPT $\leq_{\mathrm{P}}$ ACIT.

Now we prove the reverse direction. Given an arithmetic circuit C, we want to check if the polynomial $f(x_1, x_2, \ldots, x_n)$ computed by C is the zero polynomial or not. Consider the polynomial $G \stackrel{\text{def}}{=\!=} f(x_1^2, x_2^2, \ldots, x_n^2) \cdot x_1$. We know that f is non-zero iff G is non-zero. Suppose that $G \neq 0$. Now observe that in every monomial $\mathcal{M}$ of G, the degree of x_1 in $\mathcal{M}$ is odd and the degrees of the other variables $x_2, \ldots, x_n$ in $\mathcal{M}$ are even. Now consider the polynomial $H \stackrel{\text{def}}{=\!=} G(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$ where $\sigma \stackrel{\text{def}}{=\!=} (1,2)$. In every monomial $\mathcal{M}'$ of H, the degree of x_2 in $\mathcal{M}'$ is odd and the degrees of the other variables $x_1, x_3, \ldots, x_n$ in $\mathcal{M}'$ are even. Thus $H \neq G$. Hence if G is non-zero then G is not symmetric because it is not invariant under the permutation $\sigma \stackrel{\text{def}}{=\!=} (1,2)$. Thus G is symmetric iff $f = 0$. Hence ACIT $\leq_{\mathrm{P}}$ SPT. □

In contrast to SFT, SPT has a randomized polynomial time algorithm by using Lemma 6.2 (because ACIT $\in$ BPP). We have seen that SPT and SFT have contrasting computational complexities. That is, checking the symmetries in the Boolean setting seems harder than in the algebraic setting. It is natural to study this contrast in the computation of symmetric functions and polynomials.

6.2 Computing Symmetric functions and Polynomials

6.2.1 Symmetric functions

First we prove the well known result that symmetric Boolean functions (the corresponding language) are in complexity class TC^0.

Lemma 6.3. *If L is a language such that L_n is a symmetric function for all $n \in \mathbb{N}^+$, then $L \in \mathsf{TC}^0$.*

Proof. Since L_n is symmetric, we conclude that L_n only depends on the number of 1's in the input. Let us use $|x|_1$ to denote the number of 1's in $x \in \{0,1\}^n$. Let $I \subseteq [n]$ be the set such that the following is true:

$$L_n(x) = \begin{cases} 1 & |x|_1 \in I \\ 0 & \text{otherwise} \end{cases}$$

Now it is easy to see the following equality for all $x \in \{0,1\}^n$:

$$L_n(x) = \bigvee_{i \in I} (T_i(x) \wedge (\neg T_{i+1}(x))). \tag{6.2}$$

Here the T_j's (in Equation (6.2)) are the threshold gates defined in Definition 2.14 in Chapter 2. The description of L_n in Equation (6.2) is obviously a TC^0 circuit. Thus $L \in \mathsf{TC}^0$. □

6.2.2 Symmetric polynomials

We saw in Lemma 6.3 that symmetric Boolean functions are *easy* to compute. Now we show that there exist symmetric polynomials which are *hard* to compute using arithmetic circuits, assuming some well known complexity conjectures. First we define the so called elementary symmetric polynomials.

Definition 6.1. The i^{th} elementary symmetric polynomial e_i^n in n variables $x_1, x_2, \ldots, x_n$ is defined as the following polynomial:

$$e_i^n \stackrel{\text{def}}{=\!=} \sum_{1 \le j_1 < j_2 < \cdots < j_i \le n} x_{j_1} \cdot x_{j_2} \cdot \cdots \cdot x_{j_i}.$$

For an arbitrary polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, we define the polynomial f_{Sym} as:

$$f_{\text{Sym}} \stackrel{\text{def}}{=\!=} f(e_1^n, e_2^n, \ldots, e_n^n). \tag{6.3}$$

Whenever n is clear from the context, we use the notation e_i to denote the i^{th} symmetric polynomial e_i^n. Note that f_{Sym} is a symmetric polynomial. So Equation (6.3) is a method to create symmetric polynomials. The fundamental theorem of symmetric polynomials states that Equation (6.3) is the only way to create symmetric polynomials.

Theorem 6.1 ([BSC17]). *If $g \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ is a symmetric polynomial then there exists a unique polynomial $f \in \mathbb{F}[y_1, y_2, \ldots, y_n]$ such that $g = f(e_1^n, e_2^n, \ldots, e_n^n)$. Moreover, $\deg(f) \le \deg(g)$.*

Theorem 6.1 states that every symmetric polynomial g can be uniquely written as f_{Sym} for some f. Thus in whatever follows, we always use the notation of kind f_{Sym} to denote a symmetric polynomial.

Suppose we are given an arithmetic circuit C to compute f. Can we convert this circuit to a circuit computing f_{Sym}? If we could compute the elementary symmetric polynomials using *small* arithmetic circuits then we could just substitute the inputs of C by e_i^n's to obtain a circuit C_{Sym} computing f_{Sym}. It is a folklore result that e_i^n's are easy to compute.

Lemma 6.4 (Folklore). *For any $n \in \mathbb{N}^+$, $L(\{e_1^n, e_2^n, \ldots, e_n^n\}) \leq O(n^2)$.*

Proof. Consider the polynomial $g(y)$ defined as:

$$g(y) \stackrel{\text{def}}{=\!=} (y + x_1)(y + x_2)\ldots(y + x_n).$$

Observe that the coefficient of y^{n-i} in $g(y)$ is e_i^n. By polynomial interpolation we also know that the coefficients of $g(y)$ are $\mathbb{F}$-linear combinations of the evaluation of $g(y)$ at n distinct points $a_1, a_2, \ldots, a_n \in \mathbb{F}$. Therefore it follows that $e_1^n, e_2^n, \ldots, e_n^n \in \langle g(a_1), g(a_2), \ldots, g(a_n)\rangle$. Each $g(a_i)$ can be computed using $2n$ arithmetic operations: n additions and n multiplications. After computing each $g(a_i)$, it takes further $2n$ arithmetic operations to compute each e_i^n. This construction is a circuit whose outputs compute the elementary symmetric polynomials. It is clear that this circuit has size $4n^2 = O(n^2)$.

Now the following Corollary 6.1 follows easily using Lemma 6.4. □

Corollary 6.1. *For any polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, $L(f_{\text{Sym}}) \leq L(f) + O(n^2)$.*

Proof. We replace the i^{th} input of the circuit C computing f by e_i^n. By Lemma 6.4, it follows that this step can be performed by using $O(n^2)$ extra arithmetic operations. Thus $L(f_{\text{Sym}}) \leq L(f) + O(n^2)$. □

The more interesting question is that if the reverse direction of Corollary 6.1 is also true, *i.e.*, can we bound $L(f)$ in terms of $L(f_{\text{Sym}})$? This question was posed and partially solved in [GST06; DSW09; LR09]. More specifically, the following theorems were proved in [GST06; DSW09; LR09].

Theorem 6.2 (Theorem 1 in [GST06]). *For any polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, $L(f) \leq \Delta(n)L(f_{\text{Sym}}) + 2$, where $\Delta(n) \leq 4^n(n!)^2$.*

Where as [GST06] showed the bound on $L(f)$ for exact computation, [LR09] demonstrated it for approximate computations.

Theorem 6.3 ([LR09]). *For any polynomial $f \in \mathbb{Q}[x_1, x_2, \ldots, x_n]$, there is an algorithm that computes the value $f(a)$ within ϵ in time $L(f_{\text{Sym}}) + \mathsf{poly}(\log ||a||, n, \log \frac{1}{\epsilon})$ for any $a \in \mathbb{Q}^n$.*

Note that Theorem 6.3 does not compute a circuit for f but only gives an algorithm to approximate the value of f at a given point. Results in [DSW09] were in a much more general setting. [DSW09] studied the in-variance under general finite matrix groups, not just under $\mathfrak{S}_n$ as we are doing here. By specializing the theorems in [DSW09] for the finite matrix group $\mathfrak{S}_n$, we get the following result.

Theorem 6.4 ([DSW09]). *For any polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, we have $L(f) \leq ((n+1)!)^6 L(f_{\text{Sym}})$.*

The upper bound in [DSW09] (Theorem 6.4) is worse than that of [GST06] (Theorem 6.2) but this is to be expected because [DSW09] solves a more general problem.

All the exact bounds on $L(f)$ above are exponential. Now we demonstrate that $L(f)$ can be polynomially bounded in terms of $L(f_{\text{Sym}})$. For this, we need the Newton's iteration in an algebraic setting. The main proof idea is much easier to demonstrate in the case of $n = 2$. Let $B(y)$ be the following uni-variate polynomial in y with coefficients in $\mathbb{C}[x_1, x_2]$:

$$B(y) \stackrel{\text{def}}{=\!=} y^2 - (x_1 + x_2)y + x_1x_2.$$

Note that the roots of $B(y)$ are x_1, x_2. Hence we have the following equalities:

$$x_1 = \frac{x_1 + x_2 + \sqrt{(x_1 + x_2)^2 - 4x_1x_2}}{2}.$$
$$x_2 = \frac{x_1 + x_2 - \sqrt{(x_1 + x_2)^2 - 4x_1x_2}}{2}.$$

Use the symbols e_1, e_2 to define the elementary symmetric polynomials: $e_1 \stackrel{\text{def}}{=\!=} (x_1 + x_2)$ and $e_2 \stackrel{\text{def}}{=\!=} x_1x_2$. Thus we have the following equalities:

$$x_1 = \frac{e_1 + \sqrt{e_1^2 - 4e_2}}{2}. \tag{6.4}$$
$$x_2 = \frac{e_1 - \sqrt{e_1^2 - 4e_2}}{2}. \tag{6.5}$$

Let $f_{\text{Sym}} \in \mathbb{C}[x_1,x_2]$ be a symmetric polynomial with $\deg(f) = d$.

If we substitute the above radical expressions (in Equation (6.4) and Equation (6.5)) for x_1 and x_2 in $f_{\text{Sym}}(x_1, x_2)$, then we obtain $f(e_1, e_2)$. But unfortunately, we can not perform these kind of substitutions in our model of computation. This is because we can not compute expressions of the form $\sqrt{e_1^2 - 4e_2}$ in arithmetic circuits.

If we use the substitution $e_2 \leftarrow e_2 - 1$ in Equation (6.4) and Equation (6.5) and then substitute x_1 and x_2 in $f_{\text{Sym}}(x_1, x_2)$, we shall obtain $f(e_1, e_2 - 1)$. The degree of $f(e_1, e_2 - 1)$ is also bounded by d. Even by using this substitution, expressions in Equation (6.4) and Equation (6.5) can not be computed by arithmetic circuits. But this

substitution allows us to use Taylor expansion on $\sqrt{e_1^2 - 4(e_2 - 1)}$ to obtain a power series in e_1, e_2. Since $f(e_1, e_2 - 1)$ has degree at most d, we only need to substitute degree truncation d of these Taylor series to obtain $f(e_1, e_2 - 1)$ (also some additional junk terms which can be removed efficiently) and subsequently use the substitution $e_2 \leftarrow e_2 + 1$ to obtain $f(e_1, e_2)$.

This method works for two variables. This method can be extended to work for at most four variables because polynomials of degree more than four are not solvable by radicals (see section 15.9 in [BML77]). To make this idea work in general, we shall substitute e_n by $e_n + (-1)^{n-1}$ and then compute degree d truncation of roots of $B(y)$ using Newton's iteration. In whatever follows, we work with the field $\mathbb{F} = \mathbb{C}$.

6.2.2.1 *Roots as power series*

Let $F(y) = F(y, u_1, u_2, \ldots, u_n) = y^n + f_1(u_1, u_2, \ldots, u_n)y^{n-1} + \ldots + f_n(u_1, u_2, \ldots, u_n)$ be a monic square-free polynomial in variables y and $u_1, u_2, \ldots, u_n$, where $f_i \in \mathbb{C}[u_1, u_2, \ldots, u_n]$. Let $A(u_1, u_2, \ldots, u_n)$ be a root of F with respect to y. The root is usually an algebraic function in $u_1, u_2, \ldots, u_n$ but not a power series. The following Lemma 6.5 formalizes a sufficient condition where roots of $F(y)$ can be expressed as power series in $u_1, u_2, \ldots, u_n$.

Lemma 6.5 (Condition A in [SK99]). *Let $F(y, u_1, u_2, \ldots, u_n)$ be square free monic with respect to y. If $F(y, 0, 0, \ldots, 0)$ has no multiple root (as a uni-variate polynomial in y) then the roots $A_i(u_1, u_2, \ldots, u_n)$ of $F(y, u_1, u_2, \ldots, u_n)$ can be expanded into power series in $u_1, u_2, \ldots, u_n$.*

As stated above, we are interested in the following special case:

$$F(y, e_1, e_2, \ldots, e_n) = y^n - e_1 y^{n-1} + \ldots + (-1)^n e_n.$$

This F is being considered as a uni-variate polynomial in y over the power series ring $\mathbb{C}[[e_1, e_2, \ldots, e_n]]$. In this case, the roots of $F(y, e_1, e_2, \ldots, e_n)$ are $x_1, x_2, \ldots, x_n$. We want to express roots of this F as power series in $e_1, e_2, \ldots, e_n$. For this purpose, we consider a slightly modified version of F. More specifically, consider:

$$F(y, e_1, e_2, \ldots, e_n) = y^n - e_1 y^{n-1} + \ldots + (-1)^n (e_n + (-1)^{n-1}). \tag{6.6}$$

Notice that $F(y, 0, 0, \ldots, 0)$ has n distinct roots, namely the n^{th} roots of unity. Thus the roots of this $F(y)$ (Equation (6.6)) can be expressed as power series in $e_1, e_2, \ldots, e_n$, this follows from Lemma 6.5. Let us record this as Corollary 6.2.

Corollary 6.2. *If F is as in Equation (6.6), then there exist n power series $A_1, A_2, \ldots, A_n \in \mathbb{C}[[e_1, e_2, \ldots, e_n]]$ such that $F(A_i) = 0$ for all $i \in [n]$.*

Now we show how to compute the degree d truncations of such roots $A_1, A_2, \ldots, A_n$. This already follows from [KT78]. We describe the algorithm and its proof of correctness here.

6.2.2.2 *Newton's Method*

Algorithm 6.18 Newton's Method

Input: A square free monic polynomial $F(y) = F(y, u_1, u_2, \ldots, u_n) \in \mathbb{C}[u_1, u_2, \ldots, u_n][y]$ with respect to y of degree n such that $F(y, 0, 0, \ldots, 0)$ has n simple roots. A positive integer d with $d = 2^\ell$ for some $\ell \in \mathbb{N}$. We assume that $A_1, A_2, \ldots, A_n \in \mathbb{C}[[u_1, u_2, \ldots, u_n]]$ are the roots of $F(y)$.

Output: Degree d truncations $A_1^{(\ell)}, A_2^{(\ell)}, \ldots, A_n^{(\ell)}$ of n power series roots $(A_1, A_2, \ldots, A_n)$ of $F(y)$, that is, $A_i^{(\ell)} \equiv A_i \bmod I^d$ with $I \stackrel{\text{def}}{=\!=} \langle u_1, u_2, \ldots, u_n \rangle$ for all $i \in [n]$.

1: $\{\alpha_1, \alpha_2, \ldots, \alpha_n\} \leftarrow$ Roots of $F(y, 0, 0, \ldots, 0)$.
2: **for** $1 \le i \le n$ **do**
3: $\quad A_i^{(0)} \leftarrow \alpha_i$.
4: $\quad$ **for** $0 \le k \le \ell - 1$ **do**
5: $\quad\quad A_i^{(k+1)} \leftarrow A_i^{(k)} - \frac{F(A_i^{(k)})}{F'(A_i^{(k)})}$.
6: $\quad$ **end for**
7: **end for**
8: **return** $A_1^{(\ell)}, A_2^{(\ell)}, \ldots, A_n^{(\ell)}$.

For the analysis of Algorithm 6.18, define the ideal I as:

$$I \stackrel{\text{def}}{=\!=} \langle u_1, u_2, \ldots, u_n \rangle.$$

Theorem 6.5. *In Algorithm 6.18, $A_i^{(k)} \equiv A_i \bmod I^{2^k}$ for all $0 \le k \le \ell$, for all $i \in [n]$.*

Proof. Our claim is obviously true for $k = 0$. We prove the theorem by induction on k. We prove it simultaneously for all $i \in [n]$, so for the sake of brevity we use $A^{(k)}$ to denote $A_i^{(k)}$ and α to denote α_i. Consider the following equalities for a root $A = A_i$ of $F(y)$:

$$\begin{aligned} 0 = F(A) &= F(A^{(k)} + (A - A^{(k)})) \\ &= F(A^{(k)}) + (A - A^{(k)})F'(A^{(k)}) + \sum_{j>1} \frac{(A - A^{(k)})^j}{j!} F^{(j)}(A^{(k)}). \end{aligned} \tag{6.7}$$

Here $F^{(j)} \stackrel{\text{def}}{=\!=} \frac{\partial^j F(y,u_1,u_2,\ldots,u_n)}{\partial y^j}$ is the j^{th} derivative of $F(y)$ with respect to y. Since α is a simple root of $F(y,0,0,\ldots,0)$, we know that:

$$\text{Constant term of } F'(A^{(k)}) = F'(\alpha) \neq 0$$

Thus $F'(A_k)$ is inveritble in the ring $\mathbb{C}[[u_1, u_2, \ldots, u_n]]$ of power series. Therefore we have:

$$A - A^{(k+1)} = A - \left(A^{(k)} - \frac{F(A^{(k)})}{F'(A^{(k)})} \right) =$$

$$A - A^{(k+1)} = -\sum_{i>1} \frac{(A - A^{(k)})^i F^{(i)}(A^{(k)})}{i! \cdot F'(A^{(k)})}. \qquad \text{(By using Equation (6.7))}$$

Since $A - A^{(k)} \in I^{2^k}$, the right hand side of the above equation is in $I^{2^{k+1}}$. Thus $A^{(k+1)} \equiv A \bmod I^{2^{k+1}}$. □

In Algorithm 6.18, we need to compute the inverse of $F'(A^{(k)})$, since we want to compute $A^{(k+1)}$, it is enough to compute inverse of $F'(A^{(k)}) \bmod I^{2^{k+1}}$, this also follows from [KT78]. We describe this in Algorithm 6.19.

Algorithm 6.19 Inverse computation

Input: A circuit C computing the polynomial $g(u_1, u_2, \ldots, u_n)$ such that $g(0,0,\ldots,0) \neq 0$ and a positive integer d with $d = 2^\ell$ for some $\ell \in \mathbb{N}$.

Output: A circuit D for computing a polynomial $p(u_1, u_2, \ldots, u_n)$ such that $p \equiv g^{-1} \bmod I^d$, here $I = \langle u_1, u_2, \ldots, u_n \rangle$ and g^{-1} is the inverse of g in $\mathbb{C}[[u_1, u_2, \ldots, u_n]]$.

1: $p_0 \leftarrow \frac{1}{g(0,0,\ldots,0)}$.
2: **for** $0 \leq k \leq \ell - 1$ **do**
3: $\quad p_{k+1} \leftarrow p_k(2 - g \cdot p_k)$.
4: **end for**
5: **return** p_ℓ.

Lemma 6.6. *Algorithm 6.19 computes a polynomial p such that $p \equiv g^{-1} \bmod I^d$.*

Proof. We again prove it by induction on k, induction hypothesis is that $p_k \equiv g^{-1} \bmod I^{2^k}$. This induction hypothesis is trivially true for $k = 0$. Consider:

$$\begin{aligned}\frac{1}{g} - p_{k+1} &= \frac{1}{g} - p_k(2 - g \cdot p_k) \\ &= g \cdot (\frac{1}{g^2} - \frac{2p_k}{g} + p_k^2) \\ &= g \cdot (\frac{1}{g} - p_k)^2\end{aligned}$$

By using the induction hypothesis, we know that $\frac{1}{g} - p_k \in I^{2^k}$. Therefore it implies that $\frac{1}{g} - p_{k+1} \in I^{2^{k+1}}$. Now the lemma follows from the fact that $\ell = \lceil \log d \rceil$. □

We can also prove that there is a "small" circuit for p in Lemma 6.6.

Lemma 6.7. *Let $g(u_1, u_2, \ldots, u_n)$ be a polynomial such that $g(0, 0, \ldots, 0) \neq 0$. For any positive integer d with $d = 2^\ell$ for some $\ell \in \mathbb{N}$, there is a polynomial $p \in \mathbb{C}[u_1, u_2, \ldots, u_n]$ such that $p \equiv g^{-1} \bmod I^d$. Moreover, $L(p) \leq L(g) + O(\ell)$.*

Proof. In Algorithm 6.19, we need three arithmetic operations to compute p_{k+1} from p_k. It follows from Lemma 6.6 that $p_\ell = g^{-1} \bmod I^d$. Thus there exists a circuit of size $L(g) + 3 \cdot \ell = L(g) + O(\ell)$ computing $p \equiv g^{-1} \bmod I^d$. □

Now the following Theorem 6.6 follows by applying Lemma 6.7 and Theorem 6.5.

Theorem 6.6. *Let $F(y, e_1, e_2, \ldots, e_n) = y^n - e_1 y^{n-1} + \ldots + (-1)^n (e_n + (-1)^{n-1})$ and let $A_1, A_2, \ldots, A_n \in \mathbb{C}[[e_1, e_2, \ldots, e_n]]$ such that $F(A_i, e_1, e_2, \ldots, e_n) = 0$ for all $i \in [n]$. Let d be a positive integer with $d = 2^\ell$ for some $\ell \in \mathbb{N}$ and let $I = \langle e_1, e_2, \ldots, e_n \rangle$ be the ideal generated by $e_1, e_2, \ldots, e_n$ in the polynomial ring $\mathbb{C}[e_1, e_2, \ldots, e_n]$. Let polynomials D_i be such that $D_i \equiv A_i \bmod I^d$. Then $L(\{D_1, D_2, \ldots, D_n\}) \leq O(n^2 \ell + n\ell^2)$.*

Proof. We construct a circuit D whose outputs are $D_1, D_2, \ldots, D_n$. We construct the desired circuit D by using Algorithm 6.18 on $F(y, e_1, e_2, \ldots, e_n)$ and $s = (0, 0, \ldots, 0)$. It is enough to describe a circuit computing each D_i such that $D_i \equiv A_i \bmod I^d$. The circuit for $A_i^{(0)}$ in Algorithm 6.18 is trivially of size one. By line 5 of Algorithm 6.18, a circuit for $A_i^{(k+1)}$ can be constructed given any circuits of $A_i^{(k)}$, $F(A_i^{(k)})$ and $F'(A_i^{(k)})$. Note that are circuits of size $O(n)$ computing $F(y, e_1, e_2, \ldots, e_n)$ and $F'(y, e_1, e_2, \ldots, e_n) \stackrel{\text{def}}{=\!=} \frac{\partial F(y, e_1, e_2, \ldots, e_n)}{\partial y}$. Thus if $A_i^{(k)}$ has a circuit of size s then there exists a size $s + O(n + \log d)$ circuit computing $A_i^{(k+1)}$, this follows from Lemma 6.7. In particular, there exists a circuit computing $D_i \stackrel{\text{def}}{=\!=} A_i^{(\lceil \log d \rceil)}$ of size $O(n \log d + \log^2 d)$. By Theorem 6.5, it follows that $D_i \equiv A_i \bmod I^d$. We combine circuits computing D_i's to construct the desired circuit D of size $O(n \cdot n \log d + n \log^2 d) = O(n^2 \log d + n \log^2 d)$. □

Now we are ready to prove that $L(f)$ can be polynomially bounded in terms of $L(f_{\text{Sym}})$.

Theorem 6.7. *For any polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ of degree d, we have the following upper bound on $L(f)$:*

$$L(f) \leq O(d^2(L(f_{\text{Sym}}) + n^2 \log d + n \log^2 d))).$$

Proof. The main idea is what we have hinted above. Namely, let $F(y, e_1, e_2, \ldots, e_n)$ be the following polynomial:

$$F(y, e_1, e_2, \ldots, e_n) = y^n - e_1 y^{n-1} + \ldots + (-1)^n(e_n). \tag{6.8}$$

Here $e_i = e_i^n$ is the i^{th} elementary symmetric polynomial. We know that the roots (as a uni-variate polynomial in y) of F are $x_1, x_2, \ldots, x_n$. Therefore, $x_1, x_2, \ldots, x_n$ are algebraic functions in $e_1, e_2, \ldots, e_n$. Thus $x_i = A_i(e_1, e_2, \ldots, e_n)$ for some algebraic function A_i

Let $C_{\text{Sym}}(x_1, x_2, \ldots, x_n)$ be a circuit of size $L(f_{\text{Sym}})$ computing $f_{\text{Sym}}(x_1, x_2, \ldots, x_n)$. If we could substitute the x_i's by A_i's in $C_{\text{Sym}}(x_1, x_2, \ldots, x_n)$, we would obtain a circuit for f. But we cannot compute algebraic functions using arithmetic circuits. Now replace e_n by $e_n + (-1)^{n-1}$ in Equation (6.8). Thus the new $F(y, e_1, e_2, \ldots, e_n)$ is:

$$F(y, e_1, e_2, \ldots, e_n) = y^n - e_1 y^{n-1} + \ldots + (-1)^n(e_n + (-1)^{n-1}). \tag{6.9}$$

Let the roots of $F(y)$ in Equation (6.9) again be $A_1, A_2, \ldots, A_n$. By using Lemma 6.5, we know that A_i's are in $\mathbb{C}[[e_1, e_2, \ldots, e_n]]$. The following Equation (6.10) follows from the above discussion:

$$C_{\text{Sym}}(A_1, A_2, \ldots, A_n) = f(e_1, e_2, \ldots, e_n + (-1)^{n-1}). \tag{6.10}$$

To compute f, it is enough the substitute the degree d truncations of the $A_i's$ in Equation (6.10), instead of the exact infinite power series A_i. Let $D_1, D_2, \ldots, D_n$ be the outputs of circuit D obtained by applying Theorem 6.6 with degree $2^{\lceil \log d \rceil}$. We substitute the $x_i \to D_i$ in the circuit $C_{\text{Sym}}(x_1, x_2, \ldots, x_n)$. We obtain the following equality:

$$h \stackrel{\text{def}}{=\!=} C_{\text{Sym}}(D_1, D_2, \ldots, D_n) = f(e_1, e_2, \ldots, e_n + (-1)^{n-1}) + g. \tag{6.11}$$

In the above Equation (6.11), g is a polynomial with all its monomials of degree at least $d+1$, *i.e.*, $g \in I^{d+1}$ with $I = \langle e_1, e_2, \ldots, e_n \rangle$. Hence it follows that:

$$f(e_1, e_2, \ldots, e_n + (-1)^{n-1}) = \sum_{i=0}^{d} h^{[i]}.$$

By Theorem 6.6, we know that $L(h) \leq L(f_{\text{Sym}}) + (n^2 \log d + n \log^2 d)$. By using remark 2.2, we conclude that:

$$L(f(e_1, e_2, \ldots, e_n + (-1)^{n-1})) \leq O(d^2(L(f_{\text{Sym}}) + n^2 \log d + n \log^2 d))).$$

By using the substitution $e_n \to e_n - (-1)^{n-1}$, we obtain that :

$$L(f) \leq O(d^2(L(f_{\text{Sym}}) + n^2 \log d + n \log^2 d))).$$

□

Remark 6.1. In contrast to results in [GST06; DSW09], our results do depend on the degree d. But if the degree $d = \text{poly}(n)$ then our upper bound on $L(f)$ is polynomial in n and $L(f_{\text{Sym}})$. This upper bound was exponential in [GST06; DSW09].

6.2.3 Hard Symmetric Polynomials

By using the results in Subsection 6.2.2, we are ready to prove that there exist *hard* symmetric polynomials. To this end, the following Theorem 6.8 suffices.

Theorem 6.8. *Let $(f_n)_{n\in\mathbb{N}}$ be a* VNP*-complete family. Consider the corresponding symmetric polynomial family $((f_n)_{\text{Sym}})_{n\in\mathbb{N}}$. If $((f_n)_{\text{Sym}})_{n\in\mathbb{N}} \in$ VP then* VP $=$ VNP.

Proof. Suppose $f_{\text{Sym}} \overset{\text{def}}{=\!=} ((f_n)_{\text{Sym}})_{n\in\mathbb{N}}$ is in VP. Then there exists an arithmetic circuit C of size $\text{poly}(n)$ computing the symmetric polynomial $(f_n)_{\text{Sym}}$. By using Theorem 6.7, we conclude that $L(f_n) \leq \text{poly}(n)$ and thus $(f_n)_{n\in\mathbb{N}} \in$ VP. Since $(f_n)_{n\in\mathbb{N}}$ is assumed to be VNP-complete, it implies that VP $=$ VNP. □

Corollary 6.3. *Assuming* VP$\neq$ VNP*, the polynomial family $(q_n)_{n\in\mathbb{N}}$ defined by $q_n \overset{def}{=\!=} (\text{per}_n)_{\text{Sym}}$ is not in* VP.

BIBLIOGRAPHY

[AZ09] Martin Aigner and Gnter M. Ziegler. *Proofs from THE BOOK*. 4th. Springer Publishing Company, Incorporated, 2009. ISBN: 3642008550, 9783642008559.

[Ami66] S.A Amitsur. "Rational identities and applications to algebra and geometry." In: *Journal of Algebra* 3.3 (1966), pp. 304 –359. ISSN: 0021-8693. DOI: https://doi.org/10.1016/0021-8693(66)90004-4. URL: http://www.sciencedirect.com/science/article/pii/0021869366900044.

[ACR98] Alexander E. Andreev, Andrea E. F. Clementi, and José D. P. Rolim. "A New General Derandomization Method." In: *J. ACM* 45.1 (Jan. 1998), pp. 179–213. ISSN: 0004-5411. DOI: 10.1145/273865.273933. URL: http://doi.acm.org/10.1145/273865.273933.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. ISBN: 978-0-521-42426-4. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

[AL81] MD Atkinson and S Lloyd. "Primitive spaces of matrices of bounded rank." In: *Journal of the Australian Mathematical Society* 30.4 (1981), pp. 473–482.

[Bab+93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. "BPP Has Subexponential Time Simulations Unless EXPTIME has Publishable Proofs." In: *Computational Complexity* 3 (1993), pp. 307–318. DOI: 10.1007/BF01275486. URL: https://doi.org/10.1007/BF01275486.

[Bec+18] Ruben Becker, Michael Sagraloff, Vikram Sharma, and Chee Yap. "A near-optimal subdivision algorithm for complex root isolation based on the Pellet test and Newton iteration." In: *J. Symb. Comput.* 86 (2018), pp. 51–96. DOI: 10.1016/j.jsc.2017.03.009. URL: https://doi.org/10.1016/j.jsc.2017.03.009.

[BML77] Garrett Birkhoff and Saunders Mac Lane. *A survey of modern algebra*. English. 4th ed. New York : Macmillan, 1977. ISBN: 0023100702. URL: http://www.gbv.de/dms/hbz/toc/ht000038471.pdf.

[BJP18] Markus Bläser, Gorav Jindal, and Anurag Pandey. "A Deterministic PTAS for the Commutative Rank of Matrix Spaces." In: vol. 14. 3. Theory of Computing, 2018, pp. 1–21. DOI: 10.4086/toc.2018.v014a003. URL: http://www.theoryofcomputing.org/articles/v014a003.

[BSC17] Ben Blum-Smith and Samuel Coskey. "The Fundamental Theorem on Symmetric Polynomials: History's First Whiff of Galois Theory." In: *The College Mathematics Journal* 48.1 (2017), pp. 18–29. ISSN: 07468342, 19311346. URL: http://www.jstor.org/stable/10.4169/college.math.j.48.1.18.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511804441.

[BS16] Cornelius Brand and Michael Sagraloff. "On the Complexity of Solving Zero-Dimensional Polynomial Systems via Projection." In: *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*. ISSAC '16. Waterloo, ON, Canada: ACM, 2016, pp. 151–158. ISBN: 978-1-4503-4380-0. DOI: 10.1145/2930889.2930934. URL: http://doi.acm.org/10.1145/2930889.2930934.

[Bra+11] J. N. Bray, M. D. E. Conder, C. R. Leedham-Green, and E. A. O'Brien. "Short presentations for alternating and symmetric groups." In: *Trans. Amer. Math. Soc.* 363.6 (2011), pp. 3277–3285. ISSN: 0002-9947. DOI: 10.1090/S0002-9947-2011-05231-1. URL: https://doi.org/10.1090/S0002-9947-2011-05231-1.

[Büo0] Peter Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*. Springer Berlin Heidelberg, 2000. DOI: 10.1007/978-3-662-04179-6. URL: http://dx.doi.org/10.1007/978-3-662-04179-6.

[CR99] P. M. COHN and C. REUTENAUER. "ON THE CONSTRUCTION OF THE FREE FIELD." In: *International Journal of Algebra and Computation* 09.03n04 (1999), pp. 307–323. DOI: 10.1142/S0218196799000205. eprint: https://doi.org/10.1142/S0218196799000205. URL: https://doi.org/10.1142/S0218196799000205.

[CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. "The Complexity of Satisfiability of Small Depth Circuits." In: *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*. Ed. by Jianer Chen and Fedor V. Fomin. Vol. 5917. Lecture Notes in Computer Science. Springer, 2009, pp. 75–85. ISBN: 978-3-642-11268-3. DOI: 10.1007/978-3-642-11269-0_6. URL: https://doi.org/10.1007/978-3-642-11269-0_6.

[Coh75] P. M. Cohn. "The Word Problem for Free Fields: A Correction and an Addendum." In: *The Journal of Symbolic Logic* 40.1 (1975), pp. 69–74. ISSN: 00224812. URL: http://www.jstor.org/stable/2272273.

[CL76] George E. Collins and Rüdiger Loos. "Polynomial Real Root Isolation by Differentiation." In: *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation*. SYMSAC '76. Yorktown Heights, New York,

USA: ACM, 1976, pp. 15–25. DOI: 10.1145/800205.806319. URL: http://doi.acm.org/10.1145/800205.806319.

[Con78] John B. Conway. *Functions of One Complex Variable I*. Springer New York, 1978. DOI: 10.1007/978-1-4612-6313-5. URL: http://dx.doi.org/10.1007/978-1-4612-6313-5.

[Cooo0] Stephen Cook. "The P versus NP problem." In: *Clay Mathematical Institute; The Millennium Prize Problem*. 2000.

[Cos+05] Michel Coste, Tomás Lajous-Loaeza, Henri Lombardi, and Marie-Françoise Roy. "Generalized Budan–Fourier theorem and virtual roots." In: *Journal of Complexity* 21.4 (2005). Festschrift for the 70th Birthday of Arnold Schonhage, pp. 479 –486. ISSN: 0885-064X. DOI: https://doi.org/10.1016/j.jco.2004.11.003. URL: http://www.sciencedirect.com/science/article/pii/S0885064X05000075.

[CKS99] Felipe Cucker, Pascal Koiran, and Steve Smale. "A Polynomial Time Algorithm for Diophantine Equations in One Variable." In: *J. Symb. Comput.* 27.1 (1999), pp. 21–29. DOI: 10.1006/jsco.1998.0242. URL: https://doi.org/10.1006/jsco.1998.0242.

[DSW09] Xavier Dahan, Éric Schost, and Jie Wu. "Evaluation properties of invariant polynomials." In: *J. Symb. Comput.* 44.11 (2009), pp. 1592–1604. DOI: 10.1016/j.jsc.2008.12.002. URL: https://doi.org/10.1016/j.jsc.2008.12.002.

[DF04] D.S. Dummit and R.M. Foote. *Abstract Algebra*. Wiley, 2004. ISBN: 9780471433347. URL: https://books.google.de/books?id=KJDBQgAACAAJ.

[Dur+14] Arnaud Durand, Meena Mahajan, Guillaume Malod, Nicolas de Rugy-Altherre, and Nitin Saurabh. "Homomorphism Polynomials Complete for VP." In: *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*. Ed. by Venkatesh Raman and S. P. Suresh. Vol. 29. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 493–504. ISBN: 978-3-939897-77-4. DOI: 10.4230/LIPIcs.FSTTCS.2014.493. URL: http://drops.dagstuhl.de/opus/volltexte/2014/4866.

[EK95] Alan Edelman and Eric Kostlan. "How many zeros of a random polynomial are real?" In: *Bull. Amer. Math. Soc. (N.S.)* 32.1 (1995), pp. 1–37. ISSN: 0273-0979. DOI: 10.1090/S0273-0979-1995-00571-9. URL: https://doi.org/10.1090/S0273-0979-1995-00571-9.

[Edm67] Jack Edmonds. "Systems of distinct representatives and linear algebra." In: *J. Res. Nat. Bur. Standards Sect. B* 71B (1967), pp. 241–245. ISSN: 0160-1741.

[Edm03] Jack Edmonds. "Submodular Functions, Matroids, and Certain Polyhedra." In: *Combinatorial Optimization — Eureka, You Shrink!: Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France, March 5–9, 2001 Revised Papers*. Ed. by Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 11–26. ISBN: 978-3-540-36478-8. DOI: 10.1007/3-540-36478-1_2. URL: https://doi.org/10.1007/3-540-36478-1_2.

[Eig08] Arno Eigenwillig. "Real Root Isolation for Exact and Approximate Polynomials Using Descartes' Rule of Signs." PhD thesis. Saarland University, 2008. URL: https://books.google.de/books?id=PBkUmwEACAAJ.

[ESY06] Arno Eigenwillig, Vikram Sharma, and Chee K. Yap. "Almost Tight Recursion Tree Bounds for the Descartes Method." In: *Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation*. ISSAC '06. Genoa, Italy: ACM, 2006, pp. 71–78. ISBN: 1-59593-276-3. DOI: 10.1145/1145768.1145786. URL: http://doi.acm.org/10.1145/1145768.1145786.

[EH88] David Eisenbud and Joe Harris. "Vector spaces of matrices of low rank." In: *Advances in Mathematics* 70.2 (1988), pp. 135–155.

[Fla62] H. Flanders. "On Spaces of Linear Transformations with Bounded Rank." In: *Journal of the London Mathematical Society* s1-37.1 (1962), pp. 10–16. DOI: 10.1112/jlms/s1-37.1.10. eprint: /oup/backfile/content_public/journal/jlms/s1-37/1/10.1112/jlms/s1-37.1.10/2/s1-37-1-10.pdf. URL: http://dx.doi.org/10.1112/jlms/s1-37.1.10.

[FR04] Marc Fortin and Christophe Reutenauer. "Commutative/noncommutative rank of linear matrices and subspaces of matrices of low rank." eng. In: *Séminaire Lotharingien de Combinatoire* 52 (2004), B52f. URL: http://eudml.org/doc/125000.

[GST06] PIERRICK GAUDRY, ÉRIC SCHOST, and NICOLAS M. THIÉRY. "EVALUATION PROPERTIES OF SYMMETRIC POLYNOMIALS." In: *International Journal of Algebra and Computation* 16.03 (2006), pp. 505–523. DOI: 10.1142/S0218196706003128. eprint: https://doi.org/10.1142/S0218196706003128. URL: https://doi.org/10.1142/S0218196706003128.

[GG12] Maria Emilia Alonso Garcia and André Galligo. "A Root Isolation Algorithm for Sparse Univariate Polynomials." In: *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*. ISSAC '12. Grenoble, France: ACM, 2012, pp. 35–42. ISBN: 978-1-4503-1269-1. DOI: 10.1145/2442829.2442839. URL: http://doi.acm.org/10.1145/2442829.2442839.

[Gar+15] Ankit Garg, Leonid Gurvits, Rafael Oliveira, and Avi Wigderson. "Operator scaling: theory and applications." In: *arXiv preprint arXiv:1511.03730* (2015).

[Gar+16] Ankit Garg, Leonid Gurvits, Rafael Mendes de Oliveira, and Avi Wigderson. "A Deterministic Polynomial Time Algorithm for Non-commutative Rational Identity Testing." In: *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2016*. 2016, pp. 109–117. DOI: 10.1109/FOCS.2016.95. URL: https://doi.org/10.1109/FOCS.2016.95.

[Gol08] Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008. ISBN: 978-0-521-88473-0.

[Har89] Juris Hartmanis. *Godel, von Neumann and the P=? NP Problem*. Tech. rep. Cornell University, 1989.

[ISW99] R. Impagliazzo, R. Shaltiel, and A. Wigderson. "Near-optimal conversion of hardness into pseudo-randomness." In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*. 1999, pp. 181–190. DOI: 10.1109/SFFCS.1999.814590.

[IP01] Russell Impagliazzo and Ramamohan Paturi. "On the Complexity of k-SAT." In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 367–375. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.2000.1727. URL: http://www.sciencedirect.com/science/article/pii/S0022000000917276.

[ISW00] Russell Impagliazzo, Ronen Shaltiel, and Avi Wigderson. "Extractors and pseudo-random generators with optimal seed length." In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. ACM. 2000, pp. 1–10.

[IW97] Russell Impagliazzo and Avi Wigderson. "P = BPP if E Requires Exponential Circuits: Derandomizing the XOR Lemma." In: *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*. STOC '97. El Paso, Texas, USA: ACM, 1997, pp. 220–229. ISBN: 0-89791-888-6. DOI: 10.1145/258533.258590. URL: http://doi.acm.org/10.1145/258533.258590.

[IQS15] Gábor Ivanyos, Youming Qiao, and K. V. Subrahmanyam. "Constructive noncommutative rank computation in deterministic polynomial time over fields of arbitrary characteristics." In: *CoRR* abs/1512.03531 (2015). arXiv: 1512.03531. URL: http://arxiv.org/abs/1512.03531.

[IQS17a] Gábor Ivanyos, Youming Qiao, and K. V. Subrahmanyam. "Constructive Non-Commutative Rank Computation Is in Deterministic Polynomial Time." In: *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*. Ed. by Christos H. Papadimitriou. Vol. 67. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 55:1–55:19. ISBN: 978-3-95977-029-3. DOI: 10.4230/LIPIcs.ITCS.2017.55. URL: https://doi.org/10.4230/LIPIcs.ITCS.2017.55.

[IQS17b] Gábor Ivanyos, Youming Qiao, and K. V. Subrahmanyam. "Non-commutative Edmonds' problem and matrix semi-invariants." In: *computational complexity* 26.3 (2017), pp. 717–763. ISSN: 1420-8954. DOI: 10.1007/s00037-016-0143-x. URL: https://doi.org/10.1007/s00037-016-0143-x.

[Iva+15] Gábor Ivanyos, Marek Karpinski, Youming Qiao, and Miklos Santha. "Generalized Wong sequences and their applications to Edmonds' problems." In: *Journal of Computer and System Sciences* 81.7 (2015), pp. 1373–1386. URL: http://arxiv.org/abs/1307.6429.

[JS17] Gorav Jindal and Michael Sagraloff. "Efficiently Computing Real Roots of Sparse Polynomials." In: *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*. ISSAC '17. Kaiserslautern, Germany: ACM, 2017, pp. 229–236. ISBN: 978-1-4503-5064-8. DOI: 10.1145/3087604.3087652. URL: http://doi.acm.org/10.1145/3087604.3087652.

[Juk12] S. Jukna. *Boolean Function Complexity: Advances and Frontiers*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 2012. ISBN: 9783642245077. URL: https://books.google.de/books?id=BPdzkgEACAAJ.

[KI04] Valentine Kabanets and Russell Impagliazzo. "Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds." In: *Comput. Complex.* 13.1/2 (Dec. 2004), pp. 1–46. ISSN: 1016-3328. DOI: 10.1007/s00037-004-0182-6. URL: http://dx.doi.org/10.1007/s00037-004-0182-6.

[Kac43] M. Kac. "On the average number of real roots of a random algebraic equation." In: *Bull. Amer. Math. Soc.* 49.4 (Apr. 1943), pp. 314–320. URL: https://projecteuclid.org:443/euclid.bams/1183505112.

[KVV12] Dmitry S Kaliuzhnyi-Verbovetskyi and Victor Vinnikov. "Noncommutative rational functions, their difference-differential calculus and realizations." In: *Multidimensional Systems and Signal Processing* 1.23 (2012), pp. 49–77.

[KS15] Michael Kerber and Michael Sagraloff. "Root Refinement for Real Polynomials Using Quadratic Interval Refinement." In: *J. Comput. Appl. Math.* 280.C (May 2015), pp. 377–395. ISSN: 0377-0427. DOI: 10.1016/j.cam.2014.11.031. URL: http://dx.doi.org/10.1016/j.cam.2014.11.031.

[Koi11] Pascal Koiran. "Shallow circuits with high-powered inputs." In: *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*. Ed. by Bernard Chazelle. Tsinghua University Press, 2011, pp. 309–320. ISBN: 978-7-302-24517-9. URL: http://conference.itcs.tsinghua.edu.cn/ICS2011/content/papers/5.html.

[Koi17] Pascal Koiran. "Root Separation for Trinomials." working paper or preprint. Sept. 2017. URL: https://hal-ens-lyon.archives-ouvertes.fr/ensl-01585049.

[KT78] H. T. Kung and J. F. Traub. "All Algebraic Functions Can Be Computed Fast." In: *J. ACM* 25.2 (Apr. 1978), pp. 245–260. ISSN: 0004-5411. DOI: 10.1145/322063.322068. URL: http://doi.acm.org/10.1145/322063.322068.

[LJ99] Hendrik W. Lenstra (Jr.) "Finding Small Degree Factors of Lacunary Polynomials." In: *Number Theory in Progress* 1 (1999), pp. 267–276.

[LR09] Dick Lipton and Ken Regan. *Arithmetic Complexity and Symmetry*. 2009. URL: https://rjlipton.wordpress.com/2009/07/10/arithmetic-complexity-and-symmetry/.

[Lov79] László Lovász. "On determinants, matchings, and random algorithms." In: *FCT*. 1979, pp. 565–574.

[Mah14] Meena Mahajan. "Algebraic Complexity Classes." In: *Perspectives in Computational Complexity: The Somenath Biswas Anniversary Volume*. Ed. by Manindra Agrawal and Vikraman Arvind. Cham: Springer International Publishing, 2014, pp. 51–75. ISBN: 978-3-319-05446-9. DOI: 10.1007/978-3-319-05446-9_4. URL: https://doi.org/10.1007/978-3-319-05446-9_4.

[MV97] Meena Mahajan and V. Vinay. "Determinant: Combinatorics, Algorithms, and Complexity." In: *Chicago Journal of Theoretical Computer Science* 1997.5 (1997).

[Mah64] K. Mahler. "An inequality for the discriminant of a polynomial." In: *Michigan Math. J.* 11.3 (Sept. 1964), pp. 257–262. DOI: 10.1307/mmj/1028999140. URL: https://doi.org/10.1307/mmj/1028999140.

[Mar66] Morris Marden. *Geometry of Polynomials*. 2nd. Providence, RI: American Mathematical Society, 1966, pp. xiii+243. ISBN: 0821815032.

[Mar85] Morris Marden. "The Search for a Rolle's Theorem in the Complex Domain." In: *The American Mathematical Monthly* 92.9 (1985), pp. 643–650. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2323710.

[McN07] J.M. McNamee. *Numerical Methods for Roots of Polynomials*. Numerical Methods for Roots of Polynomials Teil 1. Elsevier, 2007. ISBN: 9780444527295. URL: https://books.google.de/books?id=FP8kAQAAIAAJ.

[MP13] J.M. McNamee and V. Pan. *Numerical Methods for Roots of Polynomials -*. Studies in Computational Mathematics Teil 2. Elsevier Science, 2013. ISBN: 9780080931432. URL: https://books.google.de/books?id=j0rY3D9fx-0C.

[MSW15] Kurt Mehlhorn, Michael Sagraloff, and Pengming Wang. "From approximate factorization to root isolation with application to cylindrical algebraic decomposition." In: *Journal of Symbolic Computation* 66 (2015), pp. 34 – 69. ISSN: 0747-7171. DOI: https://doi.org/10.1016/j.jsc.2014.02.001. URL: http://www.sciencedirect.com/science/article/pii/S0747717114000200.

[Meh+06] Kurt Mehlhorn, Arno Eigenwillig, Lutz Kettner, Werner Krandick, Susanne Schmitt, and Nicola Wolpert. "A Descartes Algorithms for Polynomials with Bit-Stream Coefficients." In: *Reliable Implementation of Real Number Algorithms: Theory and Practice*. Ed. by Peter Hertling, Christoph M. Hoffmann, Wolfram Luther, and Nathalie Revol. Dagstuhl Seminar Proceedings 06021. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. URL: http://drops.dagstuhl.de/opus/volltexte/2006/715.

[Mey00] C.D. Meyer. *Matrix Analysis and Applied Linear Algebra*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 2000. ISBN: 9780898714548. URL: https://books.google.de/books?id=Brpp1Cvzs14C.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge International Series on Parallel Computation. Cambridge University Press, 1995. ISBN: 9780521474658. URL: https://books.google.de/books?id=QKVY4mDivBEC.

[Nis91] Noam Nisan. "Lower bounds for non-commutative computation." In: *Proceedings of the twenty-third annual ACM symposium on Theory of computing*. ACM. 1991, pp. 410–418.

[NW94] Noam Nisan and Avi Wigderson. "Hardness vs randomness." In: *Journal of Computer and System Sciences* 49.2 (1994), pp. 149 –167. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(05)80043-1. URL: http://www.sciencedirect.com/science/article/pii/S0022000005800431.

[Obr63] N. Obreshkov. *Verteilung und Berechnung der Nullstellen reeller Polynome*. Hochschulbücher für Mathematik. Deutscher Verlag der Wissenschaften, 1963. URL: https://books.google.de/books?id=bjrSAAAAMAAJ.

[Obr03] N. Obreshkov. *Zeros of Polynomials*. Bulgarian academic monographs. Marin Drinov Academic Publishing House, 2003. ISBN: 9789544309374. URL: https://books.google.de/books?id=M-kZAQAAIAAJ.

[Orl08] James B Orlin. "A fast, simpler algorithm for the matroid parity problem." In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 2008, pp. 240–258.

[Pan97] Victor Y. Pan. "Solving a Polynomial Equation: Some History and Recent Progress." In: *SIAM Review* 39.2 (1997), pp. 187–220. DOI: 10.1137/S0036144595288554. eprint: https://doi.org/10.1137/S0036144595288554. URL: https://doi.org/10.1137/S0036144595288554.

[Pan02] Victor Y. Pan. "Univariate Polynomials: Nearly Optimal Algorithms for Numerical Factorization and Root-finding." In: *Journal of Symbolic Computation* 33.5 (2002), pp. 701 –733. ISSN: 0747-7171. DOI: https://doi.org/10.1006/jsco.2002.0531. URL: http://www.sciencedirect.com/science/article/pii/S0747717102905316.

[Pan+07] Victor Y. Pan, Brian Murphy, Rhys Eric Rosholt, Guoliang Qian, and Yuqing Tang. "Real Root-finding." In: *Proceedings of the 2007 International Workshop on Symbolic-numeric Computation*. SNC '07. London, Ontario, Canada: ACM, 2007, pp. 161–169. ISBN: 978-1-59593-744-5. DOI: 10.1145/1277500.1277524. URL: http://doi.acm.org/10.1145/1277500.1277524.

[RV89] Michael O Rabin and Vijay V Vazirani. "Maximum matchings in general graphs through randomization." In: *Journal of Algorithms* 10.4 (1989), pp. 557 –567. ISSN: 0196-6774. DOI: https://doi.org/10.1016/0196-6774(89)90005-9. URL: http://www.sciencedirect.com/science/article/pii/0196677489900059.

[RY05] J. Maurice Rojas and Yinyu Ye. "On Solving Univariate Sparse Polynomials in Logarithmic Time." In: *J. Complex.* 21.1 (Feb. 2005), pp. 87–110. ISSN: 0885-064X. DOI: 10.1016/j.jco.2004.03.004. URL: http://dx.doi.org/10.1016/j.jco.2004.03.004.

[RZ04] Fabrice Rouillier and Paul Zimmermann. "Efficient isolation of polynomial's real roots." In: *Journal of Computational and Applied Mathematics* 162.1 (2004). Proceedings of the International Conference on Linear Algebra and Arithmetic 2001, pp. 33 –50. ISSN: 0377-0427. DOI: https://doi.org/10.1016/j.cam.2003.08.015. URL: http://www.sciencedirect.com/science/article/pii/S0377042703007271.

[Rud04] Piotr Rudnicki. "Little Bezout theorem (factor theorem)." In: *FORMALIZED MATHEMATICS* 12 (2004), p. 2004.

[Sag14] Michael Sagraloff. "A Near-optimal Algorithm for Computing Real Roots of Sparse Polynomials." In: *ISSAC*. Kobe, Japan, 2014, pp. 359–366. ISBN: 978-1-4503-2501-1. DOI: 10.1145/2608628.2608632. URL: http://doi.acm.org/10.1145/2608628.2608632.

[SM16] Michael Sagraloff and Kurt Mehlhorn. "Computing real roots of real polynomials." In: *Journal of Symbolic Computation* 73 (2016), pp. 46 –86. ISSN: 0747-7171. DOI: http://dx.doi.org/10.1016/j.jsc.2015.03.004. URL: http://www.sciencedirect.com/science/article/pii/S0747717115000292.

[SK99] Tateaki Sasaki and Fujio Kako. "Solving multivariate algebraic equation by Hensel construction." In: *Japan Journal of Industrial and Applied Mathematics* 16.2 (1999), pp. 257–285. ISSN: 1868-937X. DOI: 10.1007/BF03167329. URL: https://doi.org/10.1007/BF03167329.

[Sch82] Arnold Schönhage. "The Fundamental Theorem of Algebra in Terms of Computational Complexity." In: (1982).

[Sch80] Jacob T Schwartz. "Fast probabilistic algorithms for verification of polynomial identities." In: *Journal of the ACM* 27.4 (1980), pp. 701–717.

[SU01] Ronen Shaltiel and Christopher Umans. "Simple extractors for all min-entropies and a new pseudo-random generator." In: *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*. IEEE. 2001, pp. 648–657.

[SY+10] Amir Shpilka, Amir Yehudayoff, et al. "Arithmetic circuits: A survey of recent results and open questions." In: *Foundations and Trends® in Theoretical Computer Science* 5.3–4 (2010), pp. 207–388.

[Smo93] Roman Smolensky. "On representations by low-degree polynomials." In: *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*. IEEE. 1993, pp. 130–138.

[Str73] Volker Strassen. "Vermeidung von divisionen." In: *Journal für die reine und angewandte Mathematik* 264 (1973), pp. 184–202.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. "Pseudorandom generators without the XOR lemma." In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 236–266.

[TUR36] AM TURING. "ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM." In: *J. of Math* 58 (1936), pp. 345–363.

[Tod91] Seinosuke Toda. "Counting problems computationally equivalent to computing the determinant." In: *Technical Report CSIM 91-07* (1991).

[Tut47] W. T. Tutte. "The Factorization of Linear Graphs." In: *Journal of the London Mathematical Society* s1-22.2 (1947), pp. 107–111. ISSN: 1469-7750. DOI: 10.1112/jlms/s1-22.2.107. URL: http://dx.doi.org/10.1112/jlms/s1-22.2.107.

[Uma03] Christopher Umans. "Pseudo-random generators for all hardnesses." In: *Journal of Computer and System Sciences* 67.2 (2003), pp. 419–440.

[Val79] L. G. Valiant. "Completeness Classes in Algebra." In: *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*. STOC '79. Atlanta, Georgia, USA: ACM, 1979, pp. 249–261. DOI: 10.1145/800135.804419. URL: http://doi.acm.org/10.1145/800135.804419.

[Vin91] V Vinay. "Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits." In: *Structure in Complexity Theory Conference, 1991., Proceedings of the Sixth Annual*. IEEE. 1991, pp. 270–284.

[Vol13] Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer Science & Business Media, 2013.

[Waloo] M. Waldschmidt. *Diophantine Approximation on Linear Algebraic Groups: Transcendence Properties of the Exponential Function in Several Variables*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2000. ISBN: 9783540667858. URL: https://books.google.de/books?id=p_dyoJVomEsC.

[Wano4] Xiaoshen Wang. "A Simple Proof of Descartes's Rule of Signs." In: *The American Mathematical Monthly* 111 (2004), pp. 525–526.

[Wig17] Avi Wigderson. "Operator scaling: theory, applications and connections." 2017. URL: http://www.computationalcomplexity.org/Archive/2017/tutorial.php.

[Yapoo] Chee Keng Yap. *Fundamental Problems of Algorithmic Algebra*. New York, NY, USA: Oxford University Press, Inc., 2000. ISBN: 0-19-512516-9.

[Zip79] Richard Zippel. "Probabilistic algorithms for sparse polynomials." In: *EUROSAM*. Ed. by Edward W. Ng. Vol. 72. Lecture Notes in Computer Science. Springer, 1979, pp. 216–226. ISBN: 3-540-09519-5. URL: http://dblp.uni-trier.de/db/conf/eurosam/eurosam1979.html#Zippel79.