Solving Systems of Polynomial Equations over Finite Fields

Bachelorarbeit von

Abcd Efghij

1234567

Betreuer: Prof. Dr. Holger Dell
Professur für Theoretische Informatik

21.04.2021

Goethe-Universität Frankfurt am Main
Institut für Informatik

Zusammenfassung

Das Entscheidungsproblem zur Lösbarkeit eines Gleichungssystems von Polynomen über dem endlichen Körper mit zwei Elementen $\mathbb{F}_2$ ist NP-vollständig. In vergangenen Jahren wurden Algorithmen entwickelt, welche exponenzielle der Laufzeit gegenüber dem Brute-Force-Ansatz mit Laufzeit $O^*(2^n)$ zeigten. Einer dieser Algorithmen, welcher von Björklund, Kaski und Williams [ICALP'19] veröffentlicht wurde, hat Laufzeit $O^*\left(2^{(1-1/(2.7d))n}\right)$, wobei d den Grad des Gleichungssystems bezeichnet. Der Algorithmus basiert auf einer Reduktion vom Entscheidungsproblem zum Zählproblem modulo 2 und einer Selbstreduktion. Wir verallgemeinern den Ansatz auf Gleichungssysteme von Polynomen über $\mathbb{F}_p$ für beliebige Primzahl p. Unsere Verallgemeinerung liefert eine Laufzeit von $O^*\left(p^{(1-1/(5.1(d-1)(p-1)))n}\right)$.

Abstract

The general problem of deciding whether a system of polynomial equations over the two-element field $\mathbb{F}_2$ admits a solution is a NP-complete. In recent years, algorithms yielding exponential speedups over the $O^*(2^n)$ time brute-force approach have been developed. One of these algorithms due to Björklund, Kaski and Williams [ICALP'19] runs in time $O^*\left(2^{(1-1/(2.7d))n}\right)$, where d is the degree of the system. It is based on a reduction from deciding whether a solution exists to counting solutions modulo 2 and a self-reduction. We generalise this approach to systems of polynomial equations over $\mathbb{F}_p$, where p is a prime. Our generalisation yields a running time of $O^*\left(p^{(1-1/(5.1(d-1)(p-1)))n}\right)$.

Erklärung gemäß Bachelor-Ordnung Informatik 2011 §25 Abs. 11

Hiermit erkläre ich, dass ich die Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe.

Frankfurt am Main, 21.04.2021
Abcd Efghij

Contents

1 Introduction

1.1 Satisfiability of Polynomial Equations

The aim of this thesis was to study the following decision problem for every fixed prime number p and every natural number $d > 1$:

System of Polynomial Equations over $\mathbb{F}_p$
Input: A collection of m polynomials $P_1, \ldots, P_m \in \mathbb{F}_p[X_1, \ldots, X_n]$ of degree at most d
Question: Does there exist an $x \in \mathbb{F}_p^n$ such that $P_1(x) = P_2(x) = \cdots = P_m(x) = 0$?

Systems of polynomial equations have been studied for centuries. Most students encounter them for the first time in linear algebra, where a portion of the curriculum is devoted to solving systems of linear equations over the real numbers using Gaussian elimination. More advanced areas of mathematics, such as algebraic geometry, were born out of the study of zero sets (so-called *varieties*) of finite collections of n-variate polynomials over the complex numbers [2]. On the discrete side of things, systems of polynomial equations over finite fields lend themselves to closer investigations. A simple reduction from 3Sat establishes that the problem of deciding whether a system of multivariate polynomial equations over the two-element field $\mathbb{F}_2$ has a solution is NP-hard [11]. Hence the decision problem can be seen as a generalisation of the Sat problem for Boolean CNF formulas [19]. In cryptography, systems of multivariate polynomial equations over $\mathbb{F}_2$ appear as *multivariate public key cryptosystems* [8]. Hence they are a frequent object of study at varying levels of generality [4, 19, 6].

This gives rise to the following questions: does a particular algorithmic idea allow one to replace the underlying field $\mathbb{F}_2$ with $\mathbb{F}_p$ for some prime p? In other words: which methods depend only on the algebraic structure of a finite field of prime order, instead of the more specific two-element Boolean algebra? Additionally, the complexity of such generalisations, with p acting as a parameter is of interest.

In this thesis we generalise an algorithm for systems of polynomial equations over $\mathbb{F}_2$ to $\mathbb{F}_p$, where p is an arbitrary prime number. In the following section, we provide an overview of the published work that our result builds on, followed by an outline of our contributions.

1.2 Related Work

A recent breakthrough in algorithms for systems of polynomial equations over finite fields was published by Lokshtanov et al. in 2017 [19]. The authors devised a randomised algorithm which yields an exponential speedup over the naïve $O^*(p^n)$ time brute-force approach. The authors combine a

probabilistic construction with fast multi-point evaluation of partially-assigned polynomials to obtain running times of $O^*\left(q^{(1-1/(200d))n}\right)$ for degree-d systems over $\mathbb{F}_q = \mathbb{F}_{p^k}$, where $q \geq 3$ satisfies the inequality $q \leq 2^{4ekd}$, and $O^*\left(2^{(1-1/5d)n}\right)$ for systems over $\mathbb{F}_2$. For $q \geq 3$ and $q \geq 2^{4ekd}$ they obtain a running time of $O^*\left(q^n \cdot (\log q/(ed))^{-n}\right)$.

Building on [19], Björklund et al. published an algorithm in 2019 which uses isolation techniques to reduce decision to counting, where the fast evaluation of partially-assigned polynomials is achieved through a self-reduction [6]. We describe the algorithm in depth in chapter 4.1. The algorithm by Lokshtanov et al. works over $\mathbb{F}_q$, for a prime power $q = p^k$, while the one by Björklund et al. is described only over $\mathbb{F}_2$. The algorithm in [6] has a running time of $O^*\left(2^{(1-1/(2.7d))n}\right)$. The algorithm by Björklund et al. was further refined by Dinur in 2021, yielding a running time of $O^*\left(2^{(1-1/(2d))n}\right)$ [9]. The improvements over the initial result by Lokshtanov et al. illustrate the reasoning behind our motivation to generalise the employed techniques from $\mathbb{F}_2$ to $\mathbb{F}_p$.

Some of the ideas in [6] have been applied in related domains in earlier publications: in [28, Section 6.2], Williams describes a satisfiability algorithm for ACC circuits. Two of its main ingredients are (1) brute-force evaluation of a subset of the inputs (Williams dubs this the *blowup* of the circuit) and (2) a *Coefficient-To-Point* algorithm, which evaluates a polynomial representation (obtained from the blowup) on all points of its input domain. Ingredient (1) is found in [6] in the form of self-reductions and (2) is realised using fast zeta transforms, which are equivalent to the coefficient-to-point algorithm described by Williams.

We now give an overview of the techniques employed in [19, 6] and in this thesis.

1.3 Our Techniques

For most of the techniques that are employed in both [6] as well as [19], generalisation is a straightforward task.

Assignments

To model the assignment of values to formal variables, the authors of [19] use the points contained in the n-dimensional vector space $\mathbb{F}_p^n$. In [6], subsets of $[n]$ are used to model (binary) assignments. In this thesis we use points of $\mathbb{F}_p^n$ to model value assignments.

Deciding Satisfiability

To decide the satisfiability of a set $S = \{P_1, \ldots, P_m\}$ containing m polynomials over $\mathbb{F}_p$ in n variables, the authors of [19] and [6] use different approaches. In [19] the authors define a polynomial

$$P_S(X) := 1 - \prod_{i=1}^{m} \left(1 - (P_i(X))^{p-1}\right)$$

which evaluates to 0 at $x \in \mathbb{F}_p^n$ if $P_1(x) = P_2(x) = \cdots = P_m(x) = 0$ and to 1 otherwise. Using this property, it can be concluded that the product

$$\prod_{x \in \mathbb{F}_p^n} P_S(x)$$

is equal to 0 if and only if the system of polynomial equations $P_1(X) = 0$, $P_2(X) = 0$, ..., $P_m(X) = 0$ is satisfiable.

In [6] the authors take a different approach: given a set $S = \{P_1, \ldots, P_m\}$ of m polynomials over $\mathbb{F}_2$ in n variables, they define a polynomial

$$F(X) = \prod_{i=1}^{m}(1 - P_i(X)).$$

It can be seen that F evaluates to 1 at $x \in \mathbb{F}_2^n$ if $P_1(x) = P_2(x) = \cdots = P_m(x) = 0$ and to 0 otherwise. Hence the sum

$$\sum_{x \in \mathbb{F}_2^n} F(x)$$

is equal to the number of solutions modulo p to the system $P_1(X) = 0$, $P_2(X) = 0$, ..., $P_m(X) = 0$. Using Valiant-Vazirani isolation techniques [27], the system can be enriched with $O(n)$ linear equations such that the enriched system is satisfied only by a single original solution with a controllable probability.

We take the approach of [6] and define the *counting polynomial* as

$$F(X) = \prod_{i=1}^{m}(1 - (P_i(X))^{p-1})$$

over $\mathbb{F}_p$. We also use Valiant-Vazirani isolation techniques to reduce from decision to counting solutions.

Partial Assignments

In [19], the search for a satisfying solution in a subspace of the entire domain for a parameter $n' < n$ is modeled by defining a polynomial

$$R(X_1, \ldots, X_{n-n'}) := \prod_{a \in \mathbb{F}_p^{n'}} P_S(X_1, \ldots, X_{n-n'}, a_1, \ldots, a_{n'}).$$

For any $x \in \mathbb{F}_p^{n-n'}$, it then holds that $R(x) = 0$ if and only if there exists some $a \in \mathbb{F}_p^{n'}$ such that $P_S(x_1, \ldots, x_{n-n'}, a_1, \ldots, a_{n'})$.

The authors of [6] partition the inputs $[n]$ into two sets $A \dot\cup B = [n]$ and define a *part of* F for some subset $Z \subset B$ as

$$\begin{aligned} F|_A^{Z \to B} &: 2^A \to \mathbb{F}_2 \\ &A \supseteq X \mapsto F(X \cup Z). \end{aligned} \tag{1.1}$$

They then define the *sum of a part of* F for some subset $Z \subseteq B$ as

$$I_{F|_A^{Z \to B}} := \sum_{X \subseteq A} F|_A^{Z \to B}(X) = \sum_{X \subseteq A} F(X \cup Z). \tag{1.2}$$

The sum $I_{F|_A^{Z \to B}}$ is equal to the number of satisfying assignments for whom the values of the variables whose indices are contained in Z are fixed to 1.

We follow the partition approach of [6] but define subspaces $D_A, D_B \subseteq \mathbb{F}_p^n$ which are given by the projections $D_A := \{x \in \mathbb{F}_p^n \ ; \ i \notin A \Rightarrow x_i = 0\}$ and $D_B := \{x \in \mathbb{F}_p^n \ ; \ i \notin B \Rightarrow x_i = 0\}$. Parts and sums of parts of F are then defined analogously to (1.1) and (1.2): a part of F for some $z \in D_B$ is given by

$$\begin{aligned} F|_A^{z \to B} &: D_A \to \mathbb{F}_p \\ &D_A \ni x \mapsto F(x + z), \end{aligned}$$

and the sum of a part is given by

$$I_{F|_A^{z \to B}} := \sum_{x \in D_A} F|_A^{z \to B}(x) = \sum_{x \in D_A} F(x + z)$$

for some $z \in D_B$.

Fast Multi-Point Evaluation And Interpolation

In [19], polynomials are evaluated using an algorithm that is based on fast matrix multiplication.

In [6] the authors make use of variants of the *fast zeta transform* for the subset lattice $(2^{[n]}, \subseteq)$ [7, Section 10.2] to map a polynomial which is represented by a subset of its evaluations to a representation which comprises all of its evaluations.

As the standard zeta transform for the subset lattice is a special case of *multivariate Vandermonde evaluation* and *interpolation*, we describe an algorithm that is based on Vandermonde evaluation and interpolation to realise the generalised map between representations over $\mathbb{F}_p$. We refer to it as the *Injection Algorithm*.

1.4 Our Contributions

Our main result is the following:

Theorem. There is an algorithm for degree-d systems of polynomial equations over $\mathbb{F}_p$ that runs in time $O^*\left(p^{(1-1/(5.1(d-1)(p-1)))n}\right)$.

For all primes $p < 40$, this is faster than the $O^*(p^{(1-1/(200d))n})$ time bound reported in [19], though presumably the bound given by Lokshtanov et al. is not tight.

Perhaps our most important auxiliary result is the *mod p trick* (Corollary 4.10). It lets us express the task of summing the evaluations of a polynomial across its entire domain as the task of computing summations of evaluations of partially assigned polynomials across their entire domain, followed by an interpolation. The mod p trick is a generalisation of equation (19) in [6]. The *Injection Algorithm* (Algorithm 3.6) is an interpolation that maps a polynomial of degree d given in *truncated representation* (i.e., evaluations at points whose 1-norm is bounded by d are known) to a *full representation* (i.e., evaluations at all points are known).

1.5 Structure of the Thesis

In chapter 2, we review the formal setting and basic results upon which our work builds, including the model of computation, finite fields, multivariate polynomials and evaluation and interpolation of polynomials using Vandermonde matrices. In chapter 3, we describe the aforementioned way of

modeling multivariate polynomials as elements of a vector space. We also discuss a method for multi-point evaluation and interpolation of multivariate polynomials. This culminates in the description of the Injection Algorithm. In chapter 4, we describe our generalisation of the algorithm published by Björklund et al. and prove a running-time bound. In chapter 5, we review our work and discuss further research questions.

2 Preliminaries

2.1 Notation

We settle on some notation to be used throughout this thesis. Some of the objects referred to here will be properly defined in the following section, which may be skipped if the reader is already familiar with the introduced concepts. For the purposes of this thesis, assume the smallest natural number to be 1, i.e. $\mathbb{N} = \{1, 2, \dots\}$. We use $\mathbb{N}_0$ to denote the set $\mathbb{N} \cup \{0\}$. Unless otherwise stated, assume throughout this thesis that $p \in \mathbb{N}$ is a fixed prime. We denote by $\mathbb{F} := \mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ the finite field with p elements. Its multiplicative group will be referred to by $\mathbb{F}^\times := (\mathbb{F}_p)^\times$. Assume every ring is commutative with 1. For $n \geq 1$, the set $[n] := \{1, \dots, n\}$ contains all positive integers up to n. For $n \in \mathbb{N}$, $\mathbb{P}_n := \mathbb{F}[X_1, \dots, X_n]$ denotes the polynomial ring in n variables over $\mathbb{F}$. For two finite sets S and T, we write T^S to denote the set of all functions whose signature is $S \to T$. For a map $f : S \to T$ and a subset $R \subseteq S$, we write $f|_R : R \to T$ to denote the *restriction of f to R*, given by $f|_R(r) = f(r)$. For $\alpha \in \{0, \dots, p-1\}^{[n]}$, we write X^α to denote the monomial $X_1^{\alpha(1)} X_2^{\alpha(2)} \cdots X_n^{\alpha(n)}$ and we call α a *multiexponent*. For a field k and a vector $v \in k^n$, we denote by v^t its transpose. More generally, for a $n \times m$ matrix $M \in k^{n \times m}$, we denote by M^t its transpose. If M is invertible, we denote by M^{-1} its inverse. We call the entries of a vector its *components*. The *length* of a vector is the number of its components. When the elements of a vector space are of length ℓ, we call the vector e_i for $0 \leq i \leq \ell - 1$ the *i-th canonical unit vector* if all of its components are 0 except for the $(i+1)$-th component and the $(i+1)$-th component is 1. For a vector space V, we denote by $\mathrm{id}_V : V \to V$ the *identity map*, given by $x \mapsto x$. For a vector $x \in \mathbb{F}_p^n$ that is an element of the n-dimensional vector space over $\mathbb{F}_p$, we denote by $\|x\|_1 := \sum_{i=1}^n x_i$ the *norm of x*, and the sum is taken over $\mathbb{Z}$.

2.2 Algebra

We assume the reader is familiar with basic notions from linear algebra such as groups, vector spaces and homomorphisms thereof, especially with matrices and determinants. Familiarity with basic ring theory (ideals, quotient rings) is assumed as well. We give definitions for a handful of concepts which we make use of in later portions of the thesis. We follow definitions given in [2] in this section. As we do not provide proofs for the statements given in this section, we refer to [2] or any other reference of choice for them.

Cyclic Groups

In order to introduce the discrete logarithm, we need to define cyclic groups.

2.1 Definition (Cyclic Group)**.** Let G be a finite group and let $m \in \mathbb{N}$. We say G is **cyclic of order**

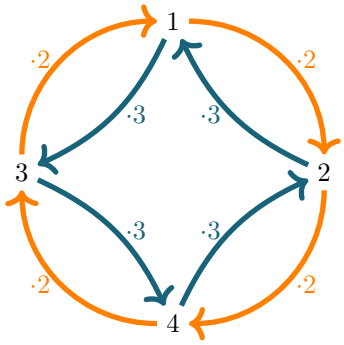

Figure 2.1: Both 2 and 3 are primitive elements of the cyclic group $(\mathbb{F}_5)^\times$. A complete visualisation of the multiplication table of $(\mathbb{F}_7)^\times$ in this fashion can be found in the appendix (A.1). This illustrates nicely why the group is called *cyclic*.

m if there exists a $\pi \in G$ such that $G = \{e, \pi, \pi^2, \ldots, \pi^{m-1}\}$, where e denotes the neutral element of the group and the powers of π are all distinct.

By definition, the elements of a cyclic group can be written as powers of an element of that group. Note that there may exist more than one such generating element. Cyclic groups allow one to define the *discrete logarithm*, which is a function that behaves similarly to the logarithm known from analysis in some respects.

Finite Fields

Recall from ring theory that all ideals of $\mathbb{Z}$ are of the form $m\mathbb{Z} = \{mz \ ; \ z \in \mathbb{Z}\}$ for some $m \in \mathbb{Z}$. For all $m \in \mathbb{Z}$, $m\mathbb{Z}$ is an ideal. Hence for all $m \in \mathbb{Z}$ the quotient ring $\mathbb{Z}/m\mathbb{Z}$ exists. For all $p \in \mathbb{N} \subset \mathbb{Z}$ which are prime, $\mathbb{Z}/p\mathbb{Z} =: \mathbb{F}_p$ is a field, called the **finite field of *p* elements**. This is often also referred to as the **Galois field of *p* elements**, named after Évariste Galois. The multiplicative group $(\mathbb{F}_p)^\times = \mathbb{F}_p \setminus \{0\}$ is cyclic of order $p-1$. If for some $\pi \in (\mathbb{F}_p)^\times$, we have $(\mathbb{F}_p)^\times = \{1, \pi, \pi^2, \ldots, \pi^{p-2}\}$, we call π a **primitive element** of $(\mathbb{F}_p)^\times$. Note again that there may be multiple primitive elements for a prime $p \in \mathbb{N}$, as for example both 2 and 3 are primitive for $p = 5$ (see Figure 2.1).

Discrete Logarithm

Let $\pi \in (\mathbb{F}_p)^\times$ be primitive. The discrete logarithm maps each element $x \in (\mathbb{F}_p)^\times$ of the cyclic group to the smallest positive integer k such that $\pi^k = x$. More formally:

2.2 Definition (Discrete Logarithm, adapted from [25])**.** The discrete logarithm with basis π, written as dlog_π is the isomorphism of groups $\mathrm{dlog}_\pi : (\mathbb{F}_p)^\times \to \mathbb{Z}/(p-1)\mathbb{Z}$.

We discard the structural part of the statement and are content with the bijection

$$(\mathbb{F}_p)^\times = \{1, 2, \ldots, p-1\} \leftrightarrow \{0, 1, \ldots, p-2\} = \mathbb{Z}/(p-1)\mathbb{Z}.$$

Fixing $\pi = 2$, we have

x	1	2	3	4
$\mathrm{dlog}_\pi(x)$	0	1	3	2

in $(\mathbb{F}_5)^\times$, to give an example.

Fermat's Little Theorem

The following abstract formulation of Fermat's Little Theorem is taken from [25].

2.3 Theorem (Fermat's Little Theorem)**.** Let G be a finite group of k elements whose neutral element we denote by e. For all $g \in G$, the following holds:

$$g^k = e.$$

Applying the theorem to $(\mathbb{F}_p)^\times$, we get

$$x^{p-1} = 1$$

for all $x \in (\mathbb{F}_p)^\times$.

Polynomials

A polynomial in the formal variable X with coefficients in a ring R (also referred to as a *polynomial in X over R*) is a formal expression of the form

$$P(X) = \sum_{i=0}^{d} a_i X^i,$$

where the a_i are elements of R. We call the a_i the *coefficients* of $P(X)$ and occasionally drop the formal variable when referring to P. When $P \neq 0$, the **degree** of P, written as $\deg(P)$ is the greatest index i such that $a_i \neq 0$. When $P = 0$, we formally set $\deg(P) = -\infty$ (see [25]). We may occasionally drop the parantheses and write $\deg P$. We refer to polynomials of degree 0 as **constant polynomials**. The set of polynomials in X over R, together with the common notions of addition and multiplication, defines a ring structure (see [2]). We refer to this ring as $R[X]$. For every $a \in R$ there is a ring homomorphism $ev_a : R[X] \to R$, which is called the **evaluation homomorphism at a**, which maps the formal variable X to a and maps the coefficients to themselves.

Seeing as $R[X]$ is a ring itself, $(R[X])[Y]$ is a valid construction. This ring contains polynomials in Y over $R[X]$. In order to describe multivariate polynomials properly, we introduce multiexponents. Since this thesis is concerned with systems of polynomial equations over finite fields, we limit our scope to multivariate polynomials over $\mathbb{F} := \mathbb{F}_p$.

2.4 Definition (Multiexponent)**.** Let $n \in \mathbb{N}$. A **multiexponent** of size n is a map $\alpha : [n] \to \{0, \ldots, p-1\} \subset \mathbb{Z}$. To each multiexponent we associate a unique monomial in n variables given by $X^\alpha = X_1^{\alpha(1)} \cdots X_n^{\alpha(n)}$. We denote by $\mathcal{M}_n$ the set of all multiexponents of size n, i.e. $\mathcal{M}_n = \{0, \ldots, p-1\}^{[n]}$.

The ring $\mathbb{F}[X_1, \ldots, X_n]$ of polynomials over $\mathbb{F}$ in n formal variables, henceforth denoted as $\mathbb{P}_n$, is inductively defined as $\mathbb{P}_0 = \mathbb{F}$ and $(\mathbb{P}_{n-1})[X_n]$ for $n > 0$ (see [25]). For a single monomial $aX_1^{k_1} X_2^{k_2} \cdots X_n^{k_n}$ with nonnegative integers k_i the **individual degree** of X_j is k_j. Since we have $x^p = x$ for any $x \in \mathbb{F}$ (see Theorem 2.3), we will assume that the highest possible individual degree of any formal variable is $p - 1$. Letting $X := (X_1, \ldots, X_n)$ denote an n-tuple of formal variables, a polynomial $P(X) \in \mathbb{P}_n$ is a linear combination of monomials:

$$P(X) = \sum_{\alpha \in \mathcal{M}_n} s_\alpha X^\alpha.$$

The s_α are *monomial coefficients* of P. The degree of a multivariate polynomial $P(X)$ with coefficients given by coefficients s_α for $\alpha \in \mathcal{M}_n$ is defined as

$$\deg(P) = \max\left\{\sum_{i=1}^{n} \alpha(i) \;;\; \alpha \in \mathcal{M}_n, s_\alpha \neq 0\right\}.$$

That is, it is defined as the highest sum of individual degrees of all monomials with non-zero coefficient. Note that the sums are taken over $\mathbb{Z}$, no matter what ring the coefficients inhabit.

Vandermonde Interpolation

Vandermonde interpolation reduces the task of evaluating a polynomial at a number of points to the computation of a matrix-vector product. Not only that, but it allows one to recover the coefficients of a polynomial, given that it has been evaluated at an appropriate selection of known points. For a ring R and $x_1, \ldots, x_m \in R$, we refer to the following $m \times m$ matrix as the *Vandermonde matrix* for $x_1, \ldots, x_m$:

$$\mathsf{Van}(x_1, \ldots, x_m) := \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{m-1} \\ \vdots & & & \ddots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^{m-1} \end{pmatrix}.$$

As described in [17, 29], the determinant of $\mathsf{Van}(x_1, \ldots, x_m)$ has been long known:

$$\det(\mathsf{Van}(x_1, \ldots, x_m)) = \prod_{i<j}(x_j - x_i),$$

hence $\mathsf{Van}(x_1, \ldots, x_m)$ is invertible if and only if the x_j are distinct. To see the fact about evaluation and interpolation alluded to as motivation, notice what happens when one multiplies $\mathsf{Van}(x_1, \ldots, x_m)$ with a vector containing the coefficients of a polynomial of degree $m-1$ in ascending order. Let $P \in R[X]$ be given by

$$P(X) = a_{m-1}X^{m-1} + a_{m-2}X^{m-2} + \cdots + a_1 X + a_0.$$

We carry out the computation of the matrix-vector product $\mathsf{Van}(x_1, \ldots, x_m) \cdot (a_0, \ldots, a_{m-1})^t$:

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{m-1} \\ \vdots & & & \ddots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^{m-1} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \end{pmatrix} = \begin{pmatrix} a_0 + a_1x_1 + a_2x_1^2 + \cdots + a_{m-1}x_1^{m-1} \\ a_0 + a_1x_2 + a_2x_2^2 + \cdots + a_{m-1}x_2^{m-1} \\ \vdots \\ a_0 + a_1x_m + a_2x_m^2 + \cdots + a_{m-1}x_m^{m-1} \end{pmatrix} = \begin{pmatrix} P(x_1) \\ P(x_2) \\ \vdots \\ P(x_n) \end{pmatrix}. \tag{2.1}$$

Thus, the matrix-vector product computes the evaluations of P at points $x_1, \ldots, x_m$. Dual to evaluation stands interpolation. If the evaluations of P at m distinct points $x_1, \ldots, x_m \in R$ are given, we can compute the coefficients $a_0, \ldots, a_{m-1}$ by inverting $\mathsf{Van}(x_1, \ldots, x_m)$ and computing the matrix-vector product $\mathsf{Van}(x_1, \ldots, x_m)^{-1} \cdot (P(x_1), \ldots, P(x_m))^t$ to get:

$$\mathsf{Van}(x_1, \ldots, x_m)^{-1} \cdot \begin{pmatrix} P(x_1) \\ \vdots \\ P(x_m) \end{pmatrix} = \mathsf{Van}(x_1, \ldots, x_m)^{-1} \cdot \left(\mathsf{Van}(x_1, \ldots, x_m) \cdot \begin{pmatrix} a_0 \\ \vdots \\ a_{m-1} \end{pmatrix} \right) = \begin{pmatrix} a_0 \\ \vdots \\ a_{m-1} \end{pmatrix},$$

using (2.1) and associativity.

Tensor Products

Tensor products allow one to turn a *multilinear* map of vector spaces $V_1 \times V_2 \times \cdots \times V_n \to W$ into a *linear* map of vector spaces $\mathcal{T} \to W$. To specify this idea, we need to define what a multilinear map is. We follow [26].

2.5 Definition (Multilinearity)**.** Let $n \in \mathbb{N}$ and let $V_1, V_2, \ldots, V_n, W$ be vector spaces of finite dimension over some field k. We call a map $f : V_1 \times V_2 \times \cdots \times V_n \to W$ **multilinear**, if for all choices of $i \in [n]$ and any fixed $s = (s_j \in V_j \ ; \ j \neq i)$, the following map of vector spaces is a homomorphism:

$$\begin{aligned} f_s : V_i &\to W \\ v &\mapsto f(s_1, \ldots, s_{i-1}, v, s_{i+1}, \ldots, s_n), \end{aligned}$$

that is, if $f_s(\lambda u + v) = \lambda f_s(u) + f_s(v)$ for all $u, v \in V_i$ and $\lambda \in k$.

In words: we say a map is multilinear if it depends linearly on each of its arguments when fixing all others [5]. The tensor product of vector spaces is often defined by means of a *universal property* [26, 3], which is a compact way of defining an object uniquely up to unique isomorphisms without imposing any structure but by determining *extrinsic* properties the object should satisfy. Since a definition in this fashion is non-constructive, the proof of existence of such an object comprises a construction.

2.6 Definition (Tensor product, translated and adapted from [26])**.** Let $n \in \mathbb{N}$ and let $V_1, V_2, \ldots, V_n$ be vector spaces of finite dimension over some field k. The **tensor product** of $V_1, \ldots, V_n$ is a vector space $V_1 \otimes \cdots \otimes V_n$ together with a multilinear map $\otimes : V_1 \times \cdots \times V_n \to V_1 \otimes \cdots \otimes V_n$ which satisfies the following property: for all finite-dimensional vector spaces W over k and all multilinear maps $f : V_1 \times \cdots \times V_n \to W$, there exists exactly one linear map $\overline{f} : V_1 \otimes \cdots \otimes V_n \to W$ such that $f = \overline{f} \circ \otimes$.

2.7 Proposition (Existence of tensor products)**.** Tensor products exist.

For the proof of Lemma 2.7 and the provided construction, we refer to the aforementioned sources. We note briefly, that if V is a vector space over k and $(a_1, \ldots, a_\ell)$ is a basis of V, then the formal elements $(b_1 \otimes b_2 \otimes \cdots \otimes b_n \ ; \ b_i \in \{a_1, \ldots, a_\ell\}$ for all $i)$ form a basis of $\bigotimes_{s=1}^{n} V =: \mathcal{T}$. Hence $\mathcal{T}$ is an ℓ^n-dimensional vector space, while V is ℓ-dimensional. For two homomorphisms $\varphi : V \to X$, $\psi : W \to Y$, there exists a homomorphism $\varphi \otimes \psi : V \otimes W \to X \otimes Y$. It is defined by $(\varphi \otimes \psi)(v \otimes w) = \varphi(v) \otimes \psi(w)$ for all $v \in V$ amd $w \in W$.

2.3 Probability

We state two inequalities which are both given in [22].

Union Bound

The following inequality holds for a collection $A_1, \ldots, A_n$ of events:

$$\Pr\left(\bigcup_{i=1}^{n} A_i\right) \leq \sum_{i=1}^{n} \Pr(A_i).$$

Note that the A_i need not necessarily be disjoint. This inequality is often referred to as the **union bound**.

Chernoff Bound

Let $X_1, \ldots, X_n$ be independent $\{0,1\}$-valued random variables and let $S := X_1 + \cdots + X_n$ denote their sum. The following inequality holds for any $\delta \in (0,1)$:

$$\Pr\left(S \leq (1-\delta)\mathbb{E}[S]\right) \leq \exp\left(-\frac{\delta^2 \mathbb{E}[S]}{2}\right).$$

2.4 Valiant-Vazirani Isolation

Valiant-Vazirani isolation techniques are the basis of our reduction from decision to counting. They were introduced in [27] in a study of the NP-complete Sat problem.

We present the matter in a fashion that is tailored to our needs, namely the task of isolating a single solution $x \in \mathcal{S}$ from a set of solutions $\mathcal{S} \subseteq \mathbb{F}_p^n$. In short, Valiant-Vazirani isolation allows us to introduce linear equations to our system of polynomial equations such that we can control the probability that only a single solution to the original system of polynomial equations satisfies the enriched system.

Pairwise Independent Hash Functions

Fix a prime $p \in \mathbb{N}$. We borrow notation from [6]. First, we introduce the notion of pairwise independent hash functions:

2.8 Definition (Pairwise Independent Hash Functions; adapted from [1])**.** Let $n, k \in \mathbb{N}$. We say a set $\mathcal{H}_{n,k}$ of functions $\mathbb{F}_p^n \to \mathbb{F}_p^k$ is **pairwise independent** if the following equation holds for all $x, x' \in \mathbb{F}_p^n$ and all $y, y' \in \mathbb{F}_p^k$ when h is uniformly distributed on $\mathcal{H}_{n,k}$:

$$\Pr_h\left(h(x) = y \wedge h(x') = y'\right) = 1 - p^{-2n}.$$

As noted in [1], pairwise independence is equivalent to the property that for any $x, x' \in \mathbb{F}_p^n$, the tuple $(h(x), h(x'))$ is uniformly distributed on $\mathbb{F}_p^k \times \mathbb{F}_p^k$ when h is chosen randomly from $\mathcal{H}_{n,k}$. For all $A \in \mathbb{F}_p^{k \times n}$ and $b \in \mathbb{F}_p^k$ we define the function

$$\begin{aligned} h_{A,b} : \mathbb{F}_p^n &\to \mathbb{F}_p^k \\ x &\mapsto Ax + b. \end{aligned}$$

The set $\mathcal{H}_{n,k} := \left\{h_{A,b} \;;\; A \in \mathbb{F}_p^{k \times n}, b \in \mathbb{F}_p^k\right\}$ is pairwise independent. We prove the property with the help of a Lemma:

2.9 Lemma. If the random variables $A_1, \ldots, A_n$ and B are independent and uniformly distributed on $\mathbb{F}_p$ and $x_1, \ldots, x_n$ are all chosen from $\mathbb{F}_p$, then the sum

$$S := B + \sum_i A_i x_i$$

is uniformly distributed on $\mathbb{F}_p$.

Proof. We distinguish between the case when all x_i are zero and the case when there is an i such that $x_i \neq 0$.

Case (i): the x_i are all zero. In this case S is equal to b, which is uniformly distributed on $\mathbb{F}_p$.

Case (ii): not all x_i are zero. Let $I \subseteq [n]$ denote the set which contains all indices i such that $x_i \neq 0$. The sum S can now be written as

$$S = B + \sum_{i \in I} A_i x_i.$$

For each $i \in I$ and each $z \in \mathbb{F}_p$, it is easy to see that $\Pr(A_i x_i = z) = \Pr(A_i = z/x_i) = 1/p$, hence $A_i x_i$ is uniformly distributed on $\mathbb{F}_p$. Note that a multiplicative inverse to x_i exists, since $x_i \neq 0$ holds for all $i \in I$.

The sum of two independent and uniformly distributed random variables $C_1, C_2 \in \mathbb{F}_p$ is uniformly distributed itself: for each $z \in \mathbb{F}_p$ we have

$$\begin{aligned}\Pr(C_1 + C_2 = z) &= \sum_{t \in \mathbb{F}_p} \Pr(C_1 = t \wedge C_2 = z - t) \\ &= \sum_{t \in \mathbb{F}_p} \Pr(C_1 = t) \cdot \Pr(C_2 = z - t) = \sum_{t \in \mathbb{F}_p} \frac{1}{p^2} = \frac{1}{p}.\end{aligned}$$

Hence S is uniformly distributed on $\mathbb{F}_p$, as it is a sum of independent uniformly distributed random variables.

□

The lemma implies that $h_{A,b}(x)$ is uniformly distributed on $\mathbb{F}_p^k$ for all $x \in \mathbb{F}_p^n$ when A is a random $k \times n$ matrix over $\mathbb{F}_p$ and b is a random vector of length k over $\mathbb{F}_p$. The matrix A maps vectors of length n to vectors of length k and the vector b accounts for the fact that the linear transformation induced by A maps the zero vector to itself (deterministically). We adapt the description and the analysis given in [6] to explain how the functions in $\mathcal{H}_{n,k}$ can be used to isolate solutions.

Isolating Solutions

Assume $\mathcal{S}$ is the set of solutions to some system of polynomial equations in n variables over $\mathbb{F}_p$. Set the natural number k such that it satisfies $p^k \leq |\mathcal{S}| < p^{k+1}$. For all $i \in \{1, \ldots, k+2\}$ and all $j \in \{1, \ldots, n\}$, let $\alpha_{i,j} \in \mathbb{F}_p$ and $\beta_j \in \mathbb{F}_p$ be independent and uniformly distributed random variables. Consider the system of linear equations in $X_1, \ldots, X_n$ given by

$$\beta_j + \sum_{i=1}^{n} \alpha_{i,j} X_i = 0$$

for $j = 1, \ldots, k+2$. Note that this system of linear equations corresponds directly to the equation $h_{A,b}(X) = 0$, where $A = (\alpha_{i,j})$ and $b = (\beta_j)^t$. We adapt the analysis from [6]: let the event S_x denote for a solution $x \in \mathcal{S}$ that x satisfies the system of linear equations and let U_x denote the event that x is the only solution which satisfies the system. We now have

$$\Pr(U_x) = \Pr\left(S_x \cap \bigcap_{y \in \mathcal{S} \setminus \{x\}} \overline{S_y}\right). \tag{2.2}$$

Lemma 2.9 implies that S_x and S_y are independent events if x is not identical to y. Hence we can rewrite (2.2) and apply a *De Morgan* law:

$$\begin{aligned}\Pr\left(S_x \cap \bigcap_{y\in\mathcal{S}\setminus\{x\}} \overline{S_y}\right) &= \Pr(S_x)\cdot\Pr\left(\bigcap_{y\in\mathcal{S}\setminus\{x\}} \overline{S_y}\right)\\ &= \Pr(S_x)\cdot\Pr\left(\overline{\bigcup_{y\in\mathcal{S}\setminus\{x\}} S_y}\right) = \Pr(S_x)\cdot\left(1-\Pr\left(\bigcup_{y\in\mathcal{S}\setminus\{x\}} S_y\right)\right).\end{aligned}$$

We apply a union bound and note that $\Pr(S_z) = 1/p^{k+2}$ holds for all $z \in \mathbb{F}_p^n$ and that $|\mathcal{S}| < p^{k+1}$:

$$\begin{aligned}\Pr(S_x)\cdot\left(1-\Pr\left(\bigcup_{y\in\mathcal{S}\setminus\{x\}} S_y\right)\right) &\geq \Pr(S_x)\cdot\left(1-\sum_{y\in\mathcal{S}\setminus\{x\}}\Pr(S_y)\right)\\ &> \frac{1}{p^{k+2}}\cdot\left(1-\frac{p^{k+1}}{p^{k+2}}\right) = \frac{p-1}{p^{k+3}} = \Pr(U_x).\end{aligned}$$

We can now give a bound on the probability that any solution $x \in \mathcal{S}$ is the only solution which satisfies the system:

$$\Pr\left(\bigcup_{x\in\mathcal{S}} U_x\right) = \sum_{x\in\mathcal{S}}\Pr(U_x) \geq p^k\cdot\frac{p-1}{p^{k+3}} = \frac{p-1}{p^3}.$$

As in [6], a total number of $\lceil \ln n \rceil$ independent repetitions isolate a single solution with probability $1 - 1/n$ for the correct choice of k, because the probability of $\lceil \ln n \rceil$ failures is

$$\left(1-\frac{p-1}{p^3}\right)^{\lceil \log n \rceil} \leq \exp\left(-\frac{p-1}{p^3}\right)^{\lceil \log n \rceil} \leq \frac{1}{n}.$$

Since the number of solutions of the system of equations is unknown to us, we try every possible value of $k \in \{0, \ldots, n\}$. This requires a total number of $O(n \log n)$ repetitions of the algorithm [6].

2.5 Model of Computation

To quote Neil Immerman: "*Turing machines have an awkward way of accessing their tapes, which makes it difficult for them to do anything in sublinear time.*" [14, §2.4]. We are not too worried about sublinear time, but the awkwardness prevails. Random access machines (RAMs) solve the awkwardness by introducing memory registers whose contents can be accessed in constant time. A prominent member of the RAM family is the *word RAM* model, which was introduced in [12] by Fredman and Willard. For a parameter $n \in \mathbb{N}$, the word RAM model operates on at most n data items stored in binary words of size b each, where $n = O(2^b)$. Standard arithmetic operations on words are modeled to require constant running time, hence the associated complexity measure is the number of arithmetic operations. A shortcoming of this model for our purposes is that the size of our input grows exponentially in our input parameter n (the number of formal variables), as we need to be given coefficients for all monomials up to a certain degree. If we let each memory cell be of size $\Theta(\log n)$, we get a model which can carry out constant-time arithmetic operations on integers whose bit-length is bounded by $\Theta(\log n)$ – a size that is agreed-upon to be reasonable [20, Section 2.2]. Yet this cell size is not adequate for our needs, as it only allows us to store $\text{poly}(n)$ data items. If we let each memory cell

be of size $\Theta(n)$ on the other hand, we venture into areas of constant-time arithmetic operations not generally agreed upon to be realistic – adding two integers of bit-length $\Theta(n)$ in constant time seems very ambitous. To settle this issue, we use the *random-access Turing machine* model.

Random-Access Turing Machines

We follow the definitions given in [15, 21] and refer to the latter for further reading. A random-access Turing machine (RTM) is a multi-tape turing machine equipped with RAM storage in the form of k *register tape pairs*, $(\mathsf{Reg}_i, \mathsf{Ram}_i)$ for $i = 1, \ldots, k$ and $k \in \mathbb{N}$. We consider RTMs with binary alphabets. Each tape is equipped with its own read/write head. The states of the Turing machine are enriched with a set of *jump states*, partitioned into the sets $J_1, \ldots, J_k$. When a state $q \in J_i$ is reached for some $1 \leq i \leq k$, the head of the tape Ram_i is moved to the cell whose address is given by the contents of the Reg_i tape, which are subsequently erased, while the Reg_i head is moved back to its starting position. We assume that these actions take a single step to complete.

In this thesis, we reserve each register tape pair for a specific purpose which is reflected by its *universe set*. For each $i \in \{1, \ldots, k\}$, we denote by Univ_i the *universe set* of RAM tape i. The universe set of RAM tape i contains the (abstract) items which are stored on the tape. Note that universe sets are not a part of the RTM definition but an abstraction that is meant easen to their analysis in our application. We take a look at an example of their use.

2.10 Example (Universe Set)**.** Consider the set $\mathsf{Univ} = \mathcal{M}_n$ which contains all multiexponents (see p.9). Let M denote some RTM with at least one register tape pair $(\mathsf{Reg}, \mathsf{Ram})$. The elements of Univ are functions $\alpha : [n] \to \{0, \ldots, p-1\}$. We choose to represent them on the RAM tape Ram in the following fashion:

$$\mathsf{bin}_p(\alpha(1))\$\mathsf{bin}_p(\alpha(2))\$ \cdots \$\mathsf{bin}_p(\alpha(n)),$$

where $\mathsf{bin}_p : \{0, \ldots, p-1\} \to \{0,1\}^{\lceil \log_2 p \rceil}$ maps integers to their base-2 representation and pads the results with leading zeroes such that all outputs have the same length. Since we assume that the degree in each variable is bounded by $p-1$, this representation requires $n \cdot \lceil \log_2 p \rceil + (n-1) = O(n)$ space to store a single multiexponent. Below is a visualisation of the representation of the multiexponent $(2, 0, 3)$ over $\mathbb{F}_5$ on the Ram tape:

⋯	0	1	0	$	0	0	0	$	0	1	1	⋯
	0	1	2	3	4	5	6	7	8	9	10	

.

The number below each cell indicates its address on the Ram tape. Note that such a representation of any 3-variate multiexponent over $\mathbb{F}_5$ requires exactly $3 \cdot \lceil \log_2 5 \rceil + 3 - 1 = 11$ cells. If $\ell > 1$ multiexponents from Univ are stored on Ram, we can therefore retrieve the i-th stored multiexponent by querying the contents of Ram in the address range $11 \cdot (i-1), \ldots, 11 \cdot i - 1$.

Taking a couple of steps back, this model suffices for reasoning about a random-access machine with an arbitrarily large address range per register and arbitrarily large memory cells. Both of these degrees of freedom are accounted for by polylogarithmic overhead resulting from having to write out addresses to register tapes. In [6], no model of computation is explicitly mentioned; the authors of [19] state in their preliminaries section that they are using the random access machine model, yet no explicit mention of word size or discussion of the point made above can be found.

Arithmetic

Since our algorithm is mostly concerned with modular arithmetic over $\mathbb{F}_p$, we briefly describe how constant-time arithmetic operations are realised. We use an addition table T_a and a multiplication table T_m, both of size $p \times p$. Their entries are given by $T_a[i,j] = i + j$ and $T_m[i,j] = i \cdot j$ for all pairs $i, j \in \mathbb{F}_p$. Both the addition table and the multiplication table can be stored on a register tape each. Again, $\mathsf{bin}_p : \{0, \ldots, p-1\} \to \{0,1\}^{\lceil \log_2 p \rceil}$ maps elements of $\mathbb{F}_p$ to a binary representation of fixed size. The entries of the addition table tape Ram_a are given by

$$(\mathsf{Ram}_a[i+jp], \mathsf{Ram}_a[i+jp+1], \ldots, \mathsf{Ram}_a[i+jp+\lceil \log_2 p \rceil - 1]) = \mathsf{bin}_p(i+j)$$

and accordingly, the entries of the multiplication table tape Ram_m are given by

$$(\mathsf{Ram}_m[i+jp], \mathsf{Ram}_m[i+jp+1], \ldots, \mathsf{Ram}_m[i+jp+\lceil \log_2 p \rceil - 1]) = \mathsf{bin}_p(i \cdot j)$$

for all $i, j \in \mathbb{F}_p$. We write $\mathsf{Ram}_i[k]$ to refer to the contents of the cell at address k of the RAM tape Ram_i. Arithmetic in $\mathbb{F}_p$ can be carried out in constant time using these tables, as p is constant.

Random Numbers

There are multiple ways to model random number generation. For our purposes, we designate a special work tape R. Let $R_1, R_2, \ldots$ be a sequence of independent random variables which are all uniformly distributed on $\{0, \ldots, p-1\} = \mathbb{F}_p$. The contents of the tape R, starting from the initial position of the tape head, are given by

$$\mathsf{bin}_p(R_1)\$\mathsf{bin}_p(R_2)\$\mathsf{bin}_p(R_3)\cdots\ ,$$

thus retrieving a random number amounts to moving the tape head of R right a total number of $\lceil \log_2 p \rceil + 1 = O(1)$ times and copying the entries at the corresponding cells.

Representing Polynomials

In the following chapter we will see that polynomials can be represented using their monomial coefficients or using their evaluations. Both representations lend themselves to a straightforward representation on RAM tapes. We sketch the representation using monomial coefficients. An n-variate polynomial $f(X) \in \mathbb{P}_n$ can be written as

$$\sum_{\alpha \in \mathcal{M}_n} s_\alpha X^\alpha,$$

where the coefficients s_α are elements of $\mathbb{F}_p$. We represent it on a RAM tape by listing all of its coefficients as we did in previous sections. The order of that listing is given by the following index function:

$$\begin{aligned} \mathsf{ind} : \mathcal{M}_n &\to \{0, \ldots, p^n - 1\} \\ \alpha &\mapsto \sum_{i=1}^{n} \alpha(i) \cdot p^{n-i}. \end{aligned}$$

We use the function bin_p to map each coefficient to a fixed-length binary representation. The address of a multiexponent can be computed on an RTM in $O(n^2)$ time, since it takes n additions of numbers whose bit-length is $O(n)$.

3 Polynomials over Finite Fields

3.1 The Vector Space $\mathbb{P}_n$

In this chapter we establish the way we use to represent multivariate polynomials as elements of a vector space. This allows us to utilise the toolbox of linear algebra to develop parts of our algorithm. To avoid expressions which may be formally correct yet unwieldy, we identify the field $\mathbb{F}_p =: \mathbb{F}$ with the set $\{0, \ldots, p-1\}$. We also identify the multiexponents $\mathcal{M}_n = \{0, \ldots, p-1\}^{[n]}$ with the elements of the vector space $\{(x_1, \ldots, x_n)^t \ ; \ x_i \in \mathbb{F}\} = \mathbb{F}^n$ through the obvious bijection $\mathcal{M}_n \ni \alpha \mapsto (\alpha(1), \ldots, \alpha(n))^t$ and its inverse $\mathbb{F}^n \ni x \mapsto \alpha_x$, where $\alpha_x(i) = x_i$ holds for all $i \in \{1, \ldots, n\}$.

Representing Polynomials

There are different ways in which a polynomial can be represented. We look at two specific ones. A multivariate polynomial $f(X) = \sum_{\alpha \in \mathcal{M}_n} s_\alpha X^\alpha \in \mathbb{P}_n$ can be given in monomial-coefficient representation:

$$\{(\alpha, s_\alpha) \ ; \ \alpha \in \mathcal{M}_n\}. \tag{3.1}$$

We will refer to this representation as *monomial representation*. Another way is the point-value-representation. It contains all evaluations of f across its entire domain:

$$\{(x, f(x)) \ ; \ x \in \mathbb{F}^n\}. \tag{3.2}$$

We will refer to this representation as *evaluation representation*.

The algebraic structure of the polynomial ring $\mathbb{P}_n$ allows for addition, subtraction and multiplication of polynomials. By allowing ourselves to lose some of that structure, we open up the possibility of using the tools of linear algebra to study multivariate polynomials: if we treat polynomials as elements of a vector space, we lose the ability to multiply them within the vector space, but we can study linear transformations of that space. The n-th "tensor power" of the vector space $\mathbb{F}^p$, written as

$$\bigotimes_{i=1}^{n} \mathbb{F}^p$$

can be seen as the natural algebraic setting for the study of n-variate polynomials. This approach enables a simple solution to a concrete problem: we know how to evaluate univariate polynomials using Vandermonde matrices (see p.10), and the tensor product view allows us to lift this approach to multivariate polynomials.

Since $x^p = x$ holds for all $x \in \mathbb{F}$, we assume that every univariate polynomial over $\mathbb{F}$ may be written in the form

$$f(X) = s_0 + s_1 X^1 + \cdots + s_{p-1} X^{p-1},$$

where the highest occuring individual degree of any formal variable is $p-1$. As an element of $\mathbb{F}^p$ (the first tensor power of $\mathbb{F}^p$), one can represent a polynomial written in this form as the linear combination

$$f = \sum_{i=0}^{p-1} s_i e_i,$$

where e_j is the j-th standard unit vector of $\mathbb{F}^p$ for $j \in \{0, \ldots, p-1\}$. To generalise this representation to multivariate polynomials, we recall what the elements of the n-th tensor power of $\mathbb{F}^p$ look like. The standard unit vectors above form a basis of $\mathbb{F}^p$, hence the formal elements $e_{i_1} \otimes \cdots \otimes e_{i_n}$ with $(i_1, \ldots, i_n) \in \{0, \ldots, p-1\}^n$, form a basis of the n-th tensor power of $\mathbb{F}^p$. We assume that every n-variate polynomial $f \in \mathbb{P}_n$ may be written in the form

$$f(X) = \sum_{\alpha \in \mathcal{M}_n} s_\alpha X^\alpha,$$

where we write X^α to abbreviate the monomial $X_1^{\alpha(1)} \cdots X_n^{\alpha(n)}$. A polynomial given in this form can be expressed as an element of the n-th tensor power as the linear combination

$$f = \sum_{\alpha \in \mathcal{M}_n} s_\alpha \left(e_{\alpha(1)} \otimes \cdots \otimes e_{\alpha(n)} \right). \tag{3.3}$$

Similarly, when given an evaluation representation, a polynomial can also be expressed as

$$\underline{f} = \sum_{x \in \mathbb{F}^n} f(x) \left(e_{x_1} \otimes \cdots \otimes e_{x_n} \right). \tag{3.4}$$

Note that we write f to denote the *monomial* representation of the polynomial $f(X)$, and we write $\underline{f}$ to denote its *evaluation* representation. We will stick to this convention throughout this chapter. We now demonstrate that this view allows us to lift the evaluation/interpolation automorphism described briefly on p.10 from the univariate case to the multivariate case.

Vandermonde mod p

The tensor product view is useful as it allows us to lift linear transformations from the univariate setting to the multivariate setting. We start with an example in the univariate setting:

3.1 Example (Univariate Vandermonde)**.** Let $f(X) = 2X + X^2 \in \mathbb{F}_3[X]$. As an element of $\mathbb{F}_3^3$, it can be written as $f = (0, 2, 1)^t$ to match the ordering of the Vandermonde matrix entries. Multiplying f with the matrix $\mathsf{Van}(0, 1, 2) \in \mathbb{F}_3^{3\times 3}$, we get the vector

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix} =: \underline{f}$$

as a result. The vector $\underline{f}$ holds the evaluations of f: $f(0) = f(1) = 0$ and $f(2) = 2$.

This example is lifted to the n-variate setting simply by applying the automorphism V_n defined by

$$V_1 := \mathsf{Van}(0, \ldots, p-1) \in \mathbb{F}^{p \times p}$$
$$V_n := V_1 \otimes V_{n-1} \in \mathbb{F}^{p^n \times p^n} \text{ for all } n > 1$$

to the tensor power representation of a polynomial.

3.2 Proposition (Multivariate Vandermonde Evaluation)**.** For all $\alpha \in \mathcal{M}_n$, the vector

$$\underline{f} := V_n \cdot \left(e_{\alpha(1)} \otimes \cdots \otimes e_{\alpha(n)}\right)$$

is equal to the point-value representation of the polynomial $f(X) = X^\alpha$. That is, we have

$$\underline{f} = \sum_{x \in \mathbb{F}^n} f(x) \left(e_{x_1} \otimes \cdots \otimes e_{x_n}\right).$$

Proof. As described in the preliminaries (p.11), applying V_n to $(e_{\alpha(1)} \otimes \cdots \otimes e_{\alpha(n)})$ for some $\alpha \in \mathcal{M}_n$ we get

$$V_n \cdot \left(e_{\alpha(1)} \otimes \cdots \otimes e_{\alpha(n)}\right) = V_1 e_{\alpha(1)} \otimes \cdots \otimes V_1 e_{\alpha(n)}. \tag{3.5}$$

Now comes a little *Indexrauferei*[1]. For each $\ell \in \{1, \ldots, n\}$, the vector $V_1 e_{\alpha(\ell)}$ contains the evaluations of the polynomial $X_\ell^{\alpha(\ell)}$ on all points of $\mathbb{F}$. It can be written as

$$V_1 e_{\alpha(\ell)} = \sum_{x_\ell=0}^{p-1} {x_\ell}^{\alpha(\ell)} \cdot e_{x_\ell}.$$

We insert this into (3.5) and apply the identities given by multilinearity:

$$\begin{aligned} V_1 e_{\alpha(1)} \otimes \cdots \otimes V_1 e_{\alpha(n)} &= \left(\sum_{x_1=0}^{p-1} {x_1}^{\alpha(1)} e_{x_1}\right) \otimes \cdots \otimes \left(\sum_{x_\ell=0}^{p-1} {x_\ell}^{\alpha(\ell)} e_{x_\ell}\right) \otimes \cdots \otimes \left(\sum_{x_n=0}^{p-1} {x_n}^{\alpha(n)} e_{x_n}\right) \\ &= \sum_{x_1=0}^{p-1} \cdots \sum_{x_n=0}^{p-1} {x_1}^{\alpha(1)} \cdots {x_n}^{\alpha(n)} \cdot \left(e_{x_1} \otimes \cdots \otimes e_{x_n}\right). \end{aligned}$$

Since the evaluation of X^α at $x = (x_1, \ldots, x_n)^t \in \mathbb{F}^n$ is given exactly by the product ${x_1}^{\alpha(1)} \cdots {x_n}^{\alpha(n)}$, this is the evaluation representation of X^α. This concludes the proof. □

Since polynomials are linear combinations of monomials, we can conclude that for any n-variate polynomial, the n-th tensor power of the Vandermonde matrix V_1 maps its monomial representation to its evaluation representation. Since the determinant of the univariate Vandermonde matrix $\mathsf{Van}(0, \ldots, p-1)$ is $\prod_{k=1}^{p-1} k!$, it is invertible. We give an explicit inverse for an arbitrary prime p in the appendix (see Section A.2).

If the map $v_i : \mathbb{F}^p \to \mathbb{F}^p$ is invertible for all $i \in \{1, \ldots, n\}$, the "tensored" map $v_1 \otimes \cdots \otimes v_n$ is invertible, with its inverse given by $v_1^{-1} \otimes \cdots \otimes v_n^{-1}$ (see p.11). Hence $V_n^{-1} := V_1^{-1} \otimes \cdots \otimes V_1^{-1}$ maps evaluation representations to monomial representations. We call this operation *multivariate Vandermonde interpolation*.

Truncated Representation

The set of n-variate polynomials whose degree is bounded by some $d \in \mathbb{N}$ is closed under addition and scalar multiplication, hence it constitutes a subspace of the vector space of all n-variate polynomials. We denote this subset by $\mathbb{P}_n^{\leq d} := \{f \in \mathbb{P}_n \ ; \ \deg f \leq d\} \subseteq \mathbb{P}_n$. We denote the vector space spanned by the basis vectors $e_{i_1} \otimes \cdots \otimes e_{i_n}$ for which $i_1 + \cdots + i_n \leq d$ holds by $\mathbb{T}_n^{\leq d}$. We identify the set of

[1] Earlier versions of this thesis included proper *Indexschlachten*, which we have since then been able to mitigate.

monomials whose degree does not exceed d with the set of points whose 1-norm does not exceed d and write them as

$$\mathcal{M}_n^{\leq d} := \{\alpha \in \mathcal{M}_n \ ; \ \deg X^\alpha \leq d\} = \{x \in \mathbb{F}^n \ ; \ \|x\|_1 \leq d\} =: \mathcal{L}_n^{\leq d}.$$

We let $\kappa_n^{\leq d}$ denote the cardinality of those sets, i.e., $\kappa_n^{\leq d} := \#\mathcal{M}_n^{\leq d} = \#\mathcal{L}_n^{\leq d}$. Common names for this quantity are *extended binomial coefficients* [10] or *compositions of d with n parts in $\{0, \ldots, p-1\}$* [13]. In [10] the authors give a condensed definition of $\kappa_n^{\leq d}$ as the sum of the cofficients of $X^0, X^1, \ldots, X^d$ in the polynomial

$$\left(1 + X^1 + \cdots + X^{p-1}\right)^n.$$

Jaklič et al. give closed-form expressions for special cases in [16] but we are content with simple bounds for our running-time analysis of the main algorithm in chapter 4.

Expressing polynomials in $\mathbb{P}_n^{\leq d}$ as elements of $\mathbb{T}_n^{\leq d}$ in *monomial* representation is trivial: as every polynomial $f(X)$ contained therein is a linear combination of monomials whose degree does not exceed d, it can be written as

$$f(X) = \sum_{\alpha \in \mathcal{M}_n^{\leq d}} s_\alpha X^\alpha.$$

The monomial representation of $f(X)$ is given by

$$f = \sum_{\alpha \in \mathcal{M}_n^{\leq d}} s_\alpha \left(e_{\alpha(1)} \otimes \cdots \otimes e_{\alpha(n)}\right).$$

Expressing polynomials from the set $\mathbb{P}_n^{\leq d}$ as elements of $\mathbb{T}_n^{\leq d}$ in *evaluation* representation on the other hand is nontrivial. Consider for example the constant polynomial $f(X) = 1$. It is contained in $\mathbb{P}_n^{\leq d}$ for all $d \in \mathbb{N}$. Its evaluation representation is

$$\underline{f} = \sum_{x \in \mathbb{F}^n} 1 \cdot \left(e_{h(x_1)} \otimes \cdots \otimes e_{h(x_n)}\right),$$

since $f(x) = 1$ for all $x \in \mathbb{F}^n$. Hence $\underline{f}$ is not an contained in $\mathbb{T}_n^{\leq d}$. Because V_n is an isomorphism and $\mathbb{T}_n^{\leq d}$ is $\kappa_n^{\leq d}$-dimensional, the image of $V_n|_{\mathbb{T}_n^{\leq d}}$ is a $\kappa_n^{\leq d}$-dimensional subspace of $\mathbb{P}_n$, but it is not necessarily $\mathbb{T}_n^{\leq d}$ itself.

We have established that a polynomial whose degree does not exceed d can be fully described in a $\kappa_n^{\leq d}$-dimensional space by listing all of its monomial coefficients. In the next section we show that this does not only hold for the monomial representation but for the evaluation representation as well. We describe an algorithm which maps a polynomial of degree d that is represented by its evaluations on all points whose 1-norm does not exceed d to all of its evaluations.

3.2 Injection

Let $f(X) = \sum_{\alpha \in \mathcal{M}_n} s_\alpha X^\alpha$ denote a polynomial whose degree is less than or equal to d. In this section we make frequent use of the following terms:

- the **monomial representation** of f contains all monomial coefficients. It can be written as the set

$$\{(\alpha, s_\alpha) \ ; \ \alpha \in \mathcal{M}_n\}$$

or equivalently as the linear combination

$$\sum_{\alpha \in \mathcal{M}_n} s_\alpha \cdot \left(e_{\alpha(1) \otimes \cdots \otimes e_{\alpha(n)}} \right).$$

- the **full evaluation representation** of f contains all point-value pairs of f. It can be written as the set

  $$\{(x, f(x)) \ ; \ x \in \mathbb{F}^n\}$$

  or equivalently as the linear combination

  $$\sum_{x \in \mathbb{F}^n} f(x) \cdot (e_{x_1} \otimes \cdots \otimes e_{x_n}).$$

- the **truncated evaluation representation** of f contains all point-value pairs of f limited to points whose 1-norm does not exceed d. It can be written as the set

  $$\left\{(x, f(x)) \ ; \ x \in \mathcal{L}_n^{\leq d}\right\}$$

  or equivalently as the linear combination

  $$\sum_{x \in \mathcal{L}_n^{\leq d}} f(x) \cdot (e_{x_1} \otimes \cdots \otimes e_{x_n}).$$

In this section we describe an algorithm which maps a polynomial given in truncated evaluation representation to its full evaluation representation. To this end, we show that the zero polynomial can be uniquely identified by its evaluations:

3.3 Lemma. Let n and d be natural numbers and let f be a polynomial whose degree is bounded by d. If $f(x) = 0$ for all $x \in \mathbb{F}^n$ which satisfy $\|x\|_1 \leq d$, then f is the zero polynomial.

Proof. We prove the statement via induction on n. For $n = 1$ and $d \leq p - 1$, we observe that the $(d+1) \times (d+1)$ Vandermonde matrix

$$V := \mathsf{Van}(0, \ldots, d) = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & d & d^2 & \cdots & d^d \end{pmatrix} \in \mathbb{F}^{(d+1) \times (d+1)}$$

has a nonzero determinant given by $\det V = \prod_{k=0}^{d} k!$, hence its kernel only contains the zero polynomial. It follows that if a univariate polynomial f whose degree does not exceed d satisfies $f(x) = 0$ for all $x \in \{0, 1, \ldots, d\}$, it must be the zero polynomial itself.

Let $n > 1$ be given such that the statement holds for $n - 1$ and choose an arbitrary $d \leq n(p-1)$. Choose any $f \in \mathbb{P}_n$ which satisfies the degree bound $\deg f \leq d$ and $f(x) = 0$ for all $x \in \mathcal{L}_n^{\leq d}$. We aim to show that $f = 0$. To this end, we write f as follows:

$$f(X_1, \ldots, X_n) = \sum_{i=0}^{p-1} f_i(X_1, \ldots, X_{n-1}) \cdot X_n^i, \tag{3.6}$$

where the f_i are $(n-1)$-variate polynomials in $X_1, \ldots, X_{n-1}$ with respective degrees $\deg f_i \leq \deg f - i$. This approach is similar to the *Coefficient-To-Point* algorithm described in [28, Section 6.2]. We iteratively show that f_{p-1}, f_{p-2}, $\ldots$, f_0 must each be zero polynomials, hence f must be a zero polynomial itself.

We set $i := p-1$ and repeatedly apply the following argument, decrementing i after each iteration until it reaches 0: choose an arbitrary $y := (y_1, \ldots, y_{n-1}) \in \mathbb{F}^{n-1}$, for which $\|y\|_1 \leq \deg f - i$ holds. Denote by $f_{\leq i}^{[y]}(X_n)$ the univariate polynomial

$$f_{\leq i}^{[y]}(X_n) := f_{\leq i}(y_1, \ldots, y_{n-1}, X_n) = \sum_{k=0}^{i} f_k(y) \cdot X_n^k,$$

where the f_k are taken from (3.6). Since f evaluates to zero on all points whose 1-norm is less than or equal to d, we have $f_{\leq i}(y_1, \ldots, y_{n-1}, x) = 0$ for all $x \in \mathbb{F}$ which satisfy $\|x\|_1 \leq i$. This is because $\deg f_{\leq i} \leq \deg f - i$ holds. Repeating the argument we used for the case $n = 1$, we can conclude that the coefficients $f_k(y)$ of $f_{\leq i}(y_1, \ldots, y_{n-1}, X_n)$ must be zero. Since this holds for all y satisfying $\|y\|_1 \leq \deg f - i$ we can conclude by the induction hypothesis that f_i must be 0.

By successive application of the argument for all $i \in \{0, \ldots, p-1\}$, we conclude that each of the f_i must be zero, hence $f = 0$. □

Lemma 3.3 yields a corollary.

3.4 Corollary. Let f be an n-variate polynomial over $\mathbb{F}$ whose degree is bounded by some $d \in \mathbb{N}$. If there is some constant $c \in \mathbb{F}$ such that $f(x) = c$ holds for all $x \in \mathcal{L}_n^{\leq d}$, then it follows that f is the constant polynomial $f(X) = c$.

Proof. Lemma 3.3 implies that the projection homomorphism

$$\mathrm{pr} : \left\{ \sum_{x \in \mathbb{F}^n} f(x) \cdot (e_{x_1} \otimes \cdots \otimes e_{x_n}) \ ; \ f \in \mathbb{P}_n^{\leq d} \right\} \to \mathbb{T}_n^{\leq d}$$

is an injective map. Recall that $\mathbb{T}_n^{\leq d}$ is the vector space

$$\mathbb{T}_n^{\leq d} = \left\{ \sum_{x \in \mathcal{L}_n^{\leq d}} c_x \cdot (e_{x_1} \otimes \cdots \otimes e_{x_n}) \ ; \ c_x \in \mathbb{F} \text{ for all } x \in \mathcal{L}_n^{\leq d} \right\}.$$

Thus we can conclude that for each $c \in \mathbb{F}$ there exists exactly one polynomial $f \in \mathbb{P}_n^{\leq d}$ which satisfies $f(x) = c$ for all $x \in \mathcal{L}_n^{\leq d}$: the constant polynomial $f(X) = c$. □

If $\underline{f}$ denotes the full evaluation representation of a polynomial $f \in \mathbb{P}_n^{\leq d}$, its truncated evaluation representation is given by $\mathrm{pr}(\underline{f}) \in \mathbb{T}_n^{\leq d}$. Corollary 3.4 is put to use in the following section: we describe an algorithm which maps truncated evaluation representations of polynomials to their full evaluation representations. Recalling how we defined those terms, our algorithm takes a description of a polynomial f in the form

$$\left\{ (x, f(x)) \ ; \ x \in \mathcal{L}_n^{\leq d} \right\}$$

and it returns a description in the form

$$\left\{ (x, f(x)) \ ; \ x \in \mathbb{F}^n \right\}.$$

Note that the possibility of existence of such an algorithm follows from Lemma 3.3 and the fact that we are considering subspaces of $\mathbb{P}_n$ which contain polynomials whose degree is upper-bounded. If we dismiss the degree bound, we lose the ability to distinguish polynomials based on their evaluations on subsets of $\mathbb{F}^n$, as the following example illustrates:

3.5 Example (Degree Restriction)**.** Consider the univariate polynomials $f(X) = 1 + X$ and $g(X) = 1 + X^2$ over $\mathbb{F}_3$. We have $f(0) = g(0) = 1$ and $f(1) = g(1) = 2$. The two polynomials cannot be distinguished based on their evaluations on the subset $\{0, 1\} \subset \mathbb{F}_3$. If we restrict ourselves to linear polynomials, we can show that there exists exactly one such polynomial h which satisfies $h(0) = 1$ and $h(1) = 2$. The inverse of the univariate Vandermonde matrix $\mathsf{Van}(0, 1, 2)$ in $\mathbb{F}_3$ is given by

$$V^{-1} := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 1 \\ 2 & 2 & 2 \end{pmatrix}.$$

The evaluation representation of h must be of the form $e := (1, 2, x)^t$ for some $x \in \mathbb{F}_3$ and its monomial representation must be of the form $m := (s_0, s_1, 0)$ for some $s_0, s_1 \in \mathbb{F}_3$. The equation $m = V^{-1}e$ admits only the solution $x = 0$:

$$\begin{aligned} & m = V^{-1}e \\ \Leftrightarrow & \begin{pmatrix} s_0 \\ s_1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 1 \\ 2 & 2 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ x \end{pmatrix} \\ \Leftrightarrow & \begin{pmatrix} s_0 \\ s_1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 + x \\ 2x \end{pmatrix}. \end{aligned}$$

Hence $h(X) = 1 + X = f(X)$ is the only linear polynomial over $\mathbb{F}_3$ which satisfies $h(0) = 1$ and $h(1) = 2$.

Because the algorithm we describe in the following section inverses the projection

$$\mathrm{pr} : \left\{ \sum_{x \in \mathbb{F}^n} f(x) \left(e_{x_1} \otimes \cdots \otimes e_{x_n}\right) \;;\; f \in \mathbb{P}_n^{\leq d} \right\} \to \mathbb{T}_n^{\leq d}$$

we refer to it as the *Injection Algorithm*.

Injection Algorithm

The main idea of the injection algorithm borrows heavily from the *Coefficient-To-Point* algorithm in [28, Section 6.2]: we can always write a polynomial $f \in \mathbb{P}_n$ as

$$f(X) = \sum_{i=0}^{d'} f_i(X_1, \ldots, X_{n-1}) \cdot X_n^i,$$

where $d' := \min\{\deg f, p-1\}$. Assume f is given in truncated evaluation representation. We recursively compute the full evaluation representations of the f_i and compute the full evaluation representation of f from those.

3.6 Algorithm (Injection)**.** Let $d \in \mathbb{N}$ be a natural number.

Input: an n-variate polynomial $f \in \mathbb{P}_n^{\leq d}$ in truncated evaluation representation (we are given $\underline{f}^{\leq d} = \sum_{x \in \mathcal{L}_n^{\leq d}} f(x)(e_{x_1} \otimes \cdots \otimes e_{x_n})$, i.e., the evaluation of f at all points whose 1-norm does not exceed d)

Output: the full evaluation representation of f (we return $\underline{f} = \sum_{x \in \mathbb{F}^n} f(x)(e_{x_1} \otimes \cdots \otimes e_{x_n})$)

Description: if f is constant on $\mathcal{L}_n^{\leq d}$, return f in full evaluation representation (**base case (a)**). If $n = 1$, perform a univariate interpolation and an evaluation using a Vandermonde matrix (**base case (b)**). Otherwise (**recursive case**), write f as

$$f(X) = \sum_{i=0}^{p-1} f_i(X_1, \ldots, X_{n-1}) \cdot X_n^i, \tag{3.7}$$

and set $d' := \min\{p-1, d\}$. Using (3.7), write

$$f_{\leq d'}(X) := \sum_{i=0}^{d'} f_i(X_1, \ldots, X_{n-1}) \cdot X_n^i.$$

Then do the following:

for $k = d'$ **to** 0 **do**

  forall $y \in \mathcal{L}_{n-1}^{\leq d-k}$ **do**

    (i) **interpolation:** interpolate the univariate polynomial $f_{\leq k}(y_1, \ldots, y_{n-1}, X_n)$ from the known evaluations $f_{\leq k}(y_1, \ldots, y_{n-1}, x)$ (for all $x \in \mathcal{L}_1^{\leq k}$) to retrieve its coefficients $f_i(y)$ for all $i \leq k$.

  end

  (ii) **recursion:** recover f_k in full evaluation representation by means of a recursive call on f_k in truncated evaluation representation (evaluations on all y obtained in previous steps)

  (iii) **degree reduction:** define the polynomial

$$f_{\leq k-1}(X) := \sum_{i=0}^{k-1} f_i(X_1, \ldots, X_{n-1}) \cdot X_n^i.$$

  forall $y \in \mathcal{L}_{n-1}^{\leq d}$ **do**

    compute the evaluation $f_{\leq k-1}(y_1, \ldots, y_{n-1}, x)$ for all $x \in \mathcal{L}_1^{\leq d-\|y\|_1}$ by noticing

$$f_{\leq k-1}(y_1, \ldots, y_{n-1}, x) = f_{\leq k}(y_1, \ldots, y_{n-1}, x) - f_k(y_1, \ldots, y_{n-1}) \cdot x^k.$$

  end

end

(iv) **aggregation:** construct the full evaluation representation of f by noticing

$$f(y_1, \ldots, y_{n-1}, x) = \sum_{i=0}^{d'} f_i(y) \cdot x^i, \tag{3.8}$$

for all $y \in \mathbb{F}^{n-1}$ and $x \in \mathbb{F}$. Since the full evaluation representations of the f_i have been computed recursively, this amounts to cycling through the values of x for each $y \in \mathbb{F}^{n-1}$ and computing the sum in (3.8).

3.7 Remark. In the description of Algorithm 3.6, the polynomials $f_{\leq k}$ are not explicitly written out as linear combinations of monomials, but rather implicitly represented this way in order to describe the interpolation and evaluation operations taking place.

We prove the correctness and provide a running time bound for the injection algorithm.

3.8 Lemma. The injection algorithm (Algorithm 3.6) is correct, i.e., for any $n \in \mathbb{N}$ and any $d \leq n(p-1)$, the algorithm maps any polynomial $f \in \mathbb{P}_n^{\leq d}$ given in truncated evaluation representation to its full evaluation representation.

3.9 Lemma. The injection algorithm (Algorithm 3.6) can be carried out by an RTM whose number of steps is bounded by $O(p^n \cdot n^2)$.

Proof of Lemma 3.8 (correctness). First we address base case (a). It is a consequence of Corollary 3.4 that the algorithm returns the correct result when it is called on a polynomial that is constant on $\mathcal{L}_n^{\leq d}$.

We now prove the correctness by induction on n for polynomials which are not constant. For $n = 1$, choose some $d \leq p - 1$ and let $f \in \mathbb{P}_1^{\leq d}$. Assume we have $f(x)$ for all $x \in \mathcal{L}_1^{\leq d}$. We first interpolate f

using the inverse of the $(d+1) \times (d+1)$ Vandermonde matrix

$$V := \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & d & d^2 & \cdots & d^d \end{pmatrix} \in \mathbb{F}^{(d+1)\times(d+1)},$$

whose determinant is $\det V = \prod_{k=0}^{d} k!$. Since the determinant of V is nonzero, f is uniquely determined and a follow-up evaluation yields its full evaluation representation (base case (b)). Therefore the induction basis holds.

Let $n > 1$ and assume the statement holds for $n-1$. Choose arbitrary $d \leq n(p-1)$ and $f \in \mathbb{P}_n^{\leq d}$. The goal of our algorithm is to compute the full evaluation basis representation of f. We write f as

$$f(X) = \sum_{i=0}^{p-1} f_i(X_1, \ldots, X_{n-1}) \cdot X_n^i.$$

We can always write f like that in a unique way, but the f_i are not neccesarily nonzero polynomials. If $d < p-1$, then $f_i = 0$ holds for all i satisfying $d < i$. Hence they don't contribute to the full evaluation basis representation of f. The degree of each constituent polynomial f_i is bounded by $d-i$. Fixing $X_n = x \in \mathbb{F}$, the full evaluation representation of $f(X_1, \ldots, X_{n-1}, x)$ is equal to the linear combination $\sum_{i=0}^{d} E_{f_i} \cdot x^i$, where E_{f_i} refers to the full evaluation representation of f_i. Hence our goal is to retrieve full representations of the f_i via recursive calls. To this end, we examine iteration k of the outer for-loop: for some $y \in \mathcal{L}_{n-1}^{\leq d-k}$, the partially evaluated $g_y(X_n) := f_{\leq k}(y_1, \ldots, y_{n-1}, X_n)$ is a univariate polynomial of degree k given by

$$g_y(X_n) = \sum_{i=0}^{k} f_i(y) \cdot X_n^i.$$

Interpolating this polynomial yields its coefficients $f_i(y)$, which are in turn evaluations of the f_i. We retrieve a truncated evaluation representation of f_k by interpolating g_y for all points y whose 1-norm is bounded by $d-k$, i.e., all $y \in \mathcal{L}_{n-1}^{\leq d-k}$. By our induction hypothesis, a recursive call on the truncated representation of f_k yields the correct full representation.

What remains to show is that for all k and $y \in \mathcal{L}_{n-1}^{\leq d-k}$, we can evaluate $g_y(X_n) = f_{\leq j}(y_1, \ldots, y_{n-1}, X_n)$ on all $x \in \mathcal{L}_1^{\leq k}$. For $k = d'$, where $d' := \min\{\deg f, p-1\}$, we have $f_{\leq k} = f$, hence the statement holds. Now assume the statement holds for some $k \leq d'$. Since $f_{\leq k-1}(X) = f_{\leq k}(X) - f_k(X_1, \ldots, X_{n-1}) \cdot X_n^k$, we use the full evaluation representation of f_k obtained in previous steps to compute the desired evaluations and conclude the proof by noting that the aggregation step described in the algorithm can be executed if the f_i are present in full evaluation representation. □

Proof of Lemma 3.9 (running time). For $n \in \mathbb{N}$, let $T(n) \in \mathbb{N}$ denote the number of steps performed on any polynomial $f \in \mathbb{P}_n$ in the worst case. We wish to show that $T(n) = O(p^n \cdot n^2)$. We first consider the recursive operations: at most p calls on $n-1$-variate polynomials are being made, making for at most $pT(n-1)$ steps from recursive calls. Now we consider the rest of the operations. The interpolations on all points in $\mathcal{L}_n^{\leq d}$ (step i) take $O(p^n \cdot n)$ steps in total, as a single univariate interpolation takes constant time and retrieving an evaluation of $f_{\leq k}$ by writing out addresses on a register tape takes $O(\log p^n) = O(n)$ steps. Note that this comprises the time required to enumerate the addresses of the RAM tapes on which the evaluations are stored. Consequently, computing the evaluations for the degree reduction (step iii) takes $O(p^n \cdot n)$ steps as well and as does the aggregation (step iv).

In total, we have the recurrence

$$T(n) = O(p^n \cdot n) + p \cdot T(n-1),$$

which is solved by $T(n) = O(p^n \cdot n^2)$. □

Putting Lemma 3.8 and Lemma 3.9 together, we can conclude:

3.10 Theorem. We can compute the full evaluation representation of a polynomial $f \in \mathbb{P}_n^{\leq d}$ given in truncated evaluation representation in $O(p^n \cdot n^2)$ time.

3.3 Enumerating Points

As the Injection Algorithm requires the enumeration of all points whose 1-norm is bounded by d, we sketch how these points can be enumerated in $O(p^n \cdot n)$ time on a random access Turing machine: we use three designated work tapes W_1, W_2 and W_3. After completion, W_3 will hold all points $x \in \mathbb{F}$ which satisfy $\|x\|_1 \leq d$, represented by

$$\mathsf{bin}_p(x_1)\$\mathsf{bin}_p(x_2)\$\cdots\$\mathsf{bin}_p(x_n), \tag{3.9}$$

where bin_p maps elements of $\mathbb{F}$ to a fixed-length binary representation (see Example 2.10). We start by writing out the point $(0, 0, \ldots, 0)$ onto the tape W_1 as in (3.9). The tape W_1 now holds all points with 1-norm 0. We copy the contents of W_1 onto W_3 and write the points $(1, 0, \ldots, 0)$, $(0, 1, 0, \ldots, 0)$, $\ldots$, $(0, \ldots, 0, 1)$ onto W_2. The tape W_2 now holds all points with 1-norm 1. We follow this up by copying the contents of W_2 to W_3 and erase the contents of W_1. For every point $x = (x_1, \ldots, x_n)$ which is encoded on W_2 we write the points $x^{(i)} = (x_1, \ldots, x_{i-1}, x_i + 1, x_{i+1}, \ldots, x_n)$ for all $i \in \{1, \ldots, n\}$ for which $x_i + 1$ is satisfied out onto W_1. The tape W_1 now holds all points whose 1-norm is 2. If we repeat these steps for d rounds, W_3 contains all points whose 1-norm is less than or equal to d. Since each point is interacted with $O(n)$ times, $O(p^n \cdot n)$ is an upper bound for the total number of steps required.

4 Solving Systems of Polynomial Equations

4.1 Description of the Algorithm

For a fixed prime p, we set $\mathbb{F} := \mathbb{F}_p$. The steps and constructions which make up our algorithm mimic those described by Björklund et al. [6] very closely in the more general $\mathbb{F}$ setting and can thus be seen as a proper generalization. Hence in the special case $p = 2$ the following sections can be seen as a rephrasing of the original algorithm in [6]. We begin with a short overview, followed by a concise decription of each step, including proofs of correctness. Note that a system of m polynomial equations given in the form

$$P_j(x) = Q_j(x) \quad \text{for all } j = 1, 2, \ldots, m$$

is equivalent to the system

$$P_j(x) - Q_j(x) = 0 \quad \text{for all } j = 1, 2, \ldots, m.$$

Thus we only consider systems of the latter, more compact form.

Overview

The problem we intend to solve is that of deciding, for a given collection of m polynomials $P_1, \ldots, P_m \in \mathbb{P}_n$ in n variables over a finite field of prime order p, whether there exists a solution of the collection, i.e., whether there exists a point $x \in \mathbb{F}^n$ which satisfies the equation $P_j(x) = 0$ for all $j = 1, 2, \ldots, m$.

By means of Valiant-Vazirani affine hashing [27], this decision problem can be reduced to that of counting the number of solutions of an appropriately modified system of polynomial equations. We put this fact to use by assuming that a solution of our system has been successfully isolated. Now we can express the task of counting solutions mod p in a compact algebraic manner: we define the n-variate polynomial

$$F(X) := (1 - (P_1(X))^{p-1}) \cdot (1 - (P_2(X))^{p-1}) \cdots (1 - (P_m(X))^{p-1}).$$

We will refer to this polynomial as the *counting polynomial* for our system. It takes on values in $\{0, 1\} \subset \mathbb{F}$ and by close inspection, one can notice its key property:

$$F(x) = 1 \quad \text{if and only if} \quad P_j(x) = 0 \text{ for all } j = 1, 2, \ldots, m.$$

This property allows us to count the solutions of our system by summing over evaluations of F across the entire domain. There are two key practical issues this approach raises:

(i) assuming our system is of degree d (i.e., $d = \max_j\{\deg P_j\}$), the counting polynomial F has degree $md(p-1)$ in the worst case

(ii) assuming our inputs are given in monomial representation, computing the representation of F can be expensive.

The solution to issue (i) comprises probabilistic constructions à la Razborov and Smolensky [23, 24]. Issue (ii) will be solved by a self-reduction which utilises the structure of the aforementioned probabilistic constructions, followed by an application of the Injection Algorithm (Algorithm 3.6). All of these steps are identical to those described in [6] on a high level. We now proceed to the in-depth description of our algorithm.

The Counting Polynomial

We start by defining an essential construction. In the following sections, we use the shorthand $X := (X_1, X_2, \ldots, X_n)$.

4.1 Definition (Counting Polynomial; adapted from [6])**.** Let $P_1(X), P_2(X), \ldots, P_m(X) \in \mathbb{P}_n$ be n-variate polynomials over the finite field of p elements. The **counting polynomial** of $P_1, P_2, \ldots, P_m$ is the n-variate polynomial

$$F(X) := \prod_{j=1}^{m} \left(1 - (P_j(X))^{p-1}\right) \in \mathbb{P}_n. \tag{4.1}$$

By Fermat's Little Theorem, we have $(P_j(x))^{p-1} = 1$ if $P_j(x) \neq 0$. Since $P_j(x) = 0$ implies $(P_j(x))^{p-1} = 0$, we have $F(x) = 1$ if and only if x is a solution of our system of polynomial equations. A similar idea is employed in [19]. The number of solutions is now given by

$$I_F := \sum_{x \in \mathbb{F}_p^n} F(x),$$

where I_f denotes the sum across the domain of a function f, as it does in [6]. Note that the degree of the counting polynomial can only be upper-bounded by $md(p-1)$ for a system of degree d. Since this is not feasible, we approximate the counting polynomial.

Approximating the Counting Polynomial

We describe a method of approximating the counting polynomial using a probabilistic polynomial of lower degree. This method of construction was first described independently by both Razborov and Smolensky in the context of circuit complexity lower bounds [23, 24]. To this end, we let $\varepsilon \in [0, 1]$ denote a parameter which we are going to set later (see p.31) and describe a probabilistic polynomial that yields the correct output (i.e., the same output as the counting polynomial) with probability at least $1 - \varepsilon$.

For $i = 1, 2, \ldots, \lceil \log_p \varepsilon^{-1} \rceil$ and $j = 1, 2, \ldots, m$ we let $\rho_{ij} \in \{0, 1, \ldots, p-1\}$ be independent uniformly distributed random variables, analogously to [6, §3.2]. For each i, we define a probabilistic polynomial

$$R_i(X) = \sum_{j=1}^{m} \rho_{ij} P_j(X).$$

Using the same principle we used for the counting polynomial, we define the *probabilistic counting polynomial* $\widetilde{F}$ as

$$\widetilde{F}(X) := \prod_{i=1}^{\lceil \log_p \varepsilon^{-1} \rceil} \left(1 - (R_i(X))^{p-1}\right) \in \mathbb{P}_n.$$

Indeed, we can control its error probability:

4.2 Lemma ([19], Lemma 3.2)**.** For any $x \in \mathbb{F}_p^n$, the following error bound holds:

$$\Pr_{\widetilde{F}}\left(F(x) = \widetilde{F}(x)\right) > 1 - \varepsilon.$$

Proof. Assume $F(x) = 1$, which means that each term in the product (4.1) has value 1 when evaluated at x. This in turn implies $P_j(x) = 0$ for all j, and thus $\widetilde{F}(x)$ takes on the value 1 deterministically.

For the converse, assume $F(x) = 0$. This means that there exists a subset $S \subseteq [n]$ of indices such that $y_s := P_s(x) \neq 0$ for all $s \in S$ and $P_t(x) = 0$ for all $t \in [n] \setminus S$. Focusing on a single probabilistic polynomial R_i, we can see that the events $\{R_i(x) = 0\}$ and $\{\sum_{s \in S} \rho_{is} \cdot y_s = 0\}$ coincide, where in the description of the second event we're summing only over the summands with potentially nonzero value. Computing the probability of this event amounts to showing that each of the $\rho_{is} y_s$ is uniformly distributed on $\{0, \ldots, p-1\}$ and that a sum of two such independent uniformly distributed random variables in is itself uniformly distributed on $\{0, \ldots, p-1\}$ (see Lemma 2.9). Hence, $\Pr(R_i(x) = 0 \mid F(x) = 0) = 1/p$ holds for all i.

Putting this together, we have $\widetilde{F}(x) = 1$ if and only if $R_i(x) = 0$ for all i. Since the ρ_{ij} are independent, we can conclude:

$$\begin{aligned}
&\Pr(\widetilde{F}(x) = 1 \mid F(x) = 0) \\
&\quad = \Pr\left(R_i = 0 \text{ for all } i = 1, 2, \ldots, \lceil \log_p \varepsilon^{-1} \rceil\right) \\
&\quad = \left(\frac{1}{p}\right)^{\lceil \log_p \varepsilon^{-1} \rceil} \leq \left(\frac{1}{p}\right)^{\log_p \varepsilon^{-1}} = p^{\log_p \varepsilon} = \varepsilon.
\end{aligned}$$

And hence $\Pr_{\widetilde{F}}(\widetilde{F}(x) = 0 \mid F(x) = 0) > 1 - \varepsilon$. □

Majority Voting Across Approximations

We split up the task of counting the solutions of the system into many tasks where we count the number of solutions under partial assignments. To this end, we partition the indices of our formal variables by setting $[n] = A \dot\cup B$. The size of the set A (and consequently also that of B) will be set in the running time analysis (see p.37). We introduce some notation:

4.3 Notation. For a subset $M \subseteq [n]$ of indices, we write D_M to denote the set $D_M := \{y \in \mathbb{F}_p^n \ ; \ i \notin M \Rightarrow y_i = 0\}$. This allows us to "compose" value assignments in the sense that the entire space of assignments $\mathbb{F}^n$ is the direct sum of D_A and D_B.

Using this notation, we formalise partial assignments:

4.4 Definition (Part of a Function; taken from [6])**.** Let $f : \mathbb{F}^n \to \mathbb{F}$ be a function and let $[n] = A \dot\cup B$. For all $z \in D_B$, we define the **part of f along z** as the function

$$\begin{aligned}
f|_A^{z \to B} : D_A &\to \mathbb{F} \\
x &\mapsto f(x + z).
\end{aligned}$$

For a part $f|_A^{z\to B}$ we refer to

$$I_{f|_A^{z\to B}} = \sum_{x\in D_A} f|_A^{z\to B}(x) = \sum_{x\in D_A} f(x+z)$$

as the sum of the part. In order to improve the probability of obtaining a correct result using our probabilistic construction, we use *scoreboarding* techniques. This means that we sample the probabilistic counting polynmial $s = O(n)$ times and take a majority vote for the sum of each part. For all $z \in D_B$ we denote by $\widetilde{I}_z^{\mathsf{maj}}$ the result of the following majority vote across the s samples:

$$\widetilde{I}_z^{\mathsf{maj}} := \mathsf{Majority}\left(I_{f_1|_A^{z\to B}}, I_{f_2|_A^{z\to B}}, \ldots, I_{f_s|_A^{z\to B}}\right),$$

for a sample $f_1, \ldots, f_s$ of s probabilistic counting polynomials. We denote the *true sum* of the part $F|_A^{z\to B}$ by I_z:

$$I_z := \sum_{x\in D_A} F|_A^{z\to B} = \sum_{x\in D_A} F(x+z),$$

and the true sum of all parts by I_F:

$$I_F := \sum_{z\in D_B} I_z.$$

By setting s to an appropriate value that is linear in n, we can provide a bound on the probability of a successful approximation of I_z using the probabilistic construction $\widetilde{I}_z^{\mathsf{maj}}$:

4.5 Lemma (Scoreboarding; adapted from [6])**.** There exists an $n \in \mathbb{N}$ which satisfies $s = O(n)$ such that

$$\Pr\left(I_F = \sum_{z\in D_B} \widetilde{I}_z^{\mathsf{maj}}\right) \geq 1 - p^{-n}$$

for all $z \in D_B$.

Proof. By Lemma 4.2, the following bound error bound holds for the evaluation of the probabilistic counting polynomial at a single point $x \in \mathbb{F}^n$:

$$\Pr_{\widetilde{F}}\left(F(x) = \widetilde{F}(x)\right) > 1 - \varepsilon.$$

The parameter ε determines the number of summands in the Razborov-Smolensky construction (see p.30). We set $\varepsilon := p^{-(|A|+2)}$ and acknowledge that the lower bound above is now given by $1 - \varepsilon = 1 - p^{-(|A|+2)}$. We fix a $z \in D_B$ and consider the probability of obtaining the correct sum of the part of F along z using the probabilistic counting polynomial. We use a *De Morgan law* and the *union bound* to simplify the expression:

$$\begin{aligned}
\Pr_{\widetilde{F}}\left(I_{F|_A^{z\to B}} = I_{\widetilde{F}|_A^{z\to B}}\right) &\geq \Pr_{\widetilde{F}}\left(\bigcap_{x\in A}\left\{F|_A^{z\to B}(x) = \widetilde{F}|_A^{z\to B}(x)\right\}\right) \\
&= 1 - \Pr_{\widetilde{F}}\left(\bigcup_{x\in A}\left\{F|_A^{z\to B}(x) \neq \widetilde{F}|_A^{z\to B}(x)\right\}\right) \\
&\geq 1 - \sum_{x\in D_A} \Pr_{\widetilde{F}}\left(F|_A^{z\to B}(x) \neq \widetilde{F}|_A^{z\to B}(x)\right) \\
&\geq 1 - \sum_{x\in D_A} p^{-(|A|+2)} = 1 - p^{|A|}\cdot p^{-(|A|+2)} = 1 - p^{-2}. \qquad (4.2)
\end{aligned}$$

Recall that $\widetilde{I}_z^{\mathsf{maj}}$ is defined as $\mathsf{Majority}\left(I_{f_1|_A^{z\to B}}, I_{f_2|_A^{z\to B}}, \ldots, I_{f_s|_A^{z\to B}}\right)$. We define a_{correct} as the number of out of the s samples of the counting polynomial for which the sum of their part along z agrees with that of the true counting polynomial:

$$a_{\mathsf{correct}} := \#\left\{j \in \{1,\ldots,s\} \;;\; I_{f_j|_A^{z\to B}} = I_{F|_A^{z\to B}}\right\}.$$

If $a_{\mathsf{correct}} > s/2$ holds, then the majority vote yields the correct result, which can be expressed as the event $\{\widetilde{I}_z^{\mathsf{maj}} = I_z\}$. Put differently, $\{a_{\mathsf{correct}} > s/2\}$ can be seen as a sufficient condition for obtaining a correct result using the scoreboarding method. By (4.2), we have $\mathbb{E}[a_{\mathsf{correct}}] \geq (1-p^{-2})s$. We examime the probability that the event $\{a_{\mathsf{correct}} > s/2\}$ occurs using a *Chernoff bound*. We set $\delta := (p^{-2} - \frac{1}{2})/(p^{-2}-1)$ and assert that $0 < \delta < 1$ since $p \geq 2$. We now have the bound

$$\begin{aligned}
\Pr_{\widetilde{F}}\left(a_{\mathsf{correct}} > \frac{s}{2}\right) &= \Pr_{\widetilde{F}}\left(a_{\mathsf{correct}} > (1-\delta)\mathbb{E}[a_{\mathsf{correct}}]\right) \\
&> 1 - \exp\left(-\frac{\delta^2 \mathbb{E}[a_{\mathsf{correct}}]}{2}\right) \\
&= 1 - \exp\left(-s\frac{(1-p^{-2})(p^{-2}-\frac{1}{2})^2}{2(p^{-2}-1)^2}\right) \\
&= 1 - \exp\left(s\frac{(p^{-2}-\frac{1}{2})^2}{2(p^{-2}-1)}\right). \qquad (4.3)
\end{aligned}$$

By setting

$$s := \left\lceil -\ln p \cdot n \cdot \frac{4(p^{-2}-1)}{(p^{-2}-\frac{1}{2})^2}\right\rceil = \left\lceil \ln p \cdot n \cdot \frac{4(1-p^{-2})}{(p^{-2}-\frac{1}{2})^2}\right\rceil = O(n)$$

and inserting it into (4.3), we get the bound

$$1 - \exp\left(s\frac{(p^{-2}-\frac{1}{2})^2}{2(p^{-2}-1)}\right) \geq 1 - \exp\left(-\ln p \cdot 2n\right) = 1 - p^{-2n}.$$

Finally, using the union bound over all $z \in D_B$, we obtain

$$\begin{aligned}
\Pr\left(I_F = \sum_{z\in D_B} \widetilde{I}_z^{\mathsf{maj}}\right) &\geq \Pr\left(\bigcap_{z\in D_B}\left\{I_z = \widetilde{I}_z^{\mathsf{maj}}\right\}\right) \\
&= 1 - \Pr\left(\bigcup_{z\in D_B}\left\{I_z \neq \widetilde{I}_z^{\mathsf{maj}}\right\}\right) \\
&\geq 1 - \sum_{z\in D_B}\Pr\left(I_z \neq \widetilde{I}_z^{\mathsf{maj}}\right) \\
&> 1 - \sum_{z\in D_B} p^{-2n} = 1 - p^{n-|A|}\cdot p^{-2n} \geq 1 - p^{-n}.
\end{aligned}$$

This concludes the proof. □

The mod p Trick

Recall from the overview that the algorithm includes a self-reduction. Our goal is to avoid having to compute an explicit representation of the probabilistic counting polynomial as a linear combination of monomials. The same approach is used in [6]. Our contribution is a restatement over $\mathbb{F}$. We start with a basic lemma before we proceed with our key result that is needed for the restatement. We have dubbed this result the *mod p trick*.

4.6 Notation. To improve legibility, we write $\lambda(r)$ to denote $\mathrm{dlog}_\pi(r)$, where π is a fixed primitive element of the multiplicative group $\mathbb{F}^\times$.

4.7 Lemma. The following statement holds for any $\alpha \in \{1, 2, \ldots, p-2\} \subset \mathbb{Z}$:

$$\sum_{r \in \mathbb{F}^\times} \left(\pi_p^{\lambda(r)}\right)^\alpha = 0. \tag{4.4}$$

Proof. Let $\alpha \in \{1, \ldots, p-2\} \subset \mathbb{Z}$. We inspect the sum (4.4) and use the fact that dlog_{π_p} is a bijection:

$$\sum_{r \in \mathbb{F}^\times} \left(\pi_p^{\lambda(r)}\right)^\alpha = \sum_{r \in \mathbb{F}^\times} \left(\pi_p^\alpha\right)^{\lambda(r)} = \sum_{i=0}^{p-2} \left(\pi_p^\alpha\right)^i = \frac{\left(\pi_p^\alpha\right)^{p-1} - 1}{\pi_p^\alpha - 1} = \frac{1-1}{\pi_p^\alpha - 1} = 0.$$

We applied the closed form for geometric sums, which we can do since $\pi_p^\alpha \neq 1$. □

We specify a special subset of the multiexponents of size n, which we refer to as the *top of* $\mathcal{M}_n$:

4.8 Definition (Top of $\mathcal{M}_n$)**.** Let $M \subseteq [n]$. We define the **top of $\mathcal{M}_n$ with respect to M** as the set

$$[\top]_M := \{\alpha \in \mathcal{M}_n \ ; \ i \in M \Rightarrow \alpha(i) = p-1\} \subseteq \mathcal{M}_n.$$

There is a bijection from $\{0, \ldots, p-1\}^{[n]\setminus M}$ to $[\top]_M$ which is given by

$$\begin{aligned} \iota : \{0, \ldots, p-1\}^{[n]\setminus M} &\to [\top]_M \\ \beta &\mapsto \iota(\beta) =: \beta', \\ \text{where } \beta'(k) &= \begin{cases} \beta(k), & \text{if } k \notin M. \\ p-1, & \text{otherwise.} \end{cases} \end{aligned}$$

One can quickly confirm that ι is indeed a bijection. This will come in handy in the next step, where we describe the self-reduction.

Let $z \in D_B$. For a polynomial $f(X) = \sum_{\alpha \in \mathcal{M}_n} s_\alpha X^\alpha$, we write the sum of the part $f|_A^{z \to B}$ using the freshly introduced notation:

$$\begin{aligned} I_{f|_A^{z\to B}} = \sum_{x \in D_A} f(x+z) &= \sum_{x \in D_A} \sum_{\alpha \in \mathcal{M}_n} s_\alpha \cdot \prod_{i \in A} x_i^{\alpha(i)} \cdot \prod_{i \in B} z_i^{\alpha(i)} \\ &= \sum_{\alpha \in \mathcal{M}_n} \left(s_\alpha \cdot \prod_{i \in B} z_i^{\alpha(i)} \cdot \sum_{x \in D_A} \prod_{i \in A} x_i^{\alpha(i)} \right) \end{aligned} \tag{4.5}$$

We state a proposition concerning the inner sum in (4.5), from which we can instantly deduce our contribution in the form of Corollary 4.10:

4.9 Proposition. For any $\alpha \in \mathcal{M}_n$ and a nonempty subset $A \subseteq [n]$, the following holds:

(i) $\alpha \in [\top]_A$ implies $\displaystyle\sum_{x \in D_A} \prod_{i \in A} x_i^{\alpha(i)} = (p-1)^{|A|} = \begin{cases} 1, & \text{if } |A| \text{ is even.} \\ p-1, & \text{if } |A| \text{ is odd.} \end{cases}$

(ii) $\alpha \notin [\top]_A$ implies $\displaystyle\sum_{x \in D_A} \prod_{i \in A} x_i^{\alpha(i)} = 0.$

4.10 Corollary (mod p trick)**.** For a polynomial $f(X) = \sum_{\alpha \in \mathcal{M}_n} s_\alpha X^\alpha$ and any $z \in D_B$, the following holds:

$$I_{f|_A^{z\to B}} = \sum_{\alpha \in [\top]_A} s_\alpha \cdot \prod_{i \in B} z_i^{\alpha(i)} \cdot (p-1)^{|A|}.$$

This corollary is motivated by the following idea: given a polynomial $f(X) = \sum_{\alpha \in \mathcal{M}_n} s_\alpha X^\alpha \in \mathbb{P}_n$, one can define a B-variate[1] polynomial $\Phi(X_B)$ with the property that its evaluation $\Phi(z) = I_{f|_A^{z \to B}}$ at $z \in D_B$ is equal to the sum of the part $I_{f|_A^{z \to B}}$. We are going to describe this technique in detail in the following sections. To conclude this section, we provide a proof of Proposition 4.9.

Proof of Proposition 4.9. For (i), let $\alpha \in [\top]_A$ and choose an arbitrary $x \in D_A$. We notice that

$$\prod_{i \in A} x_i^{\alpha(i)} = \prod_{i \in A} x_i^{p-1} = \begin{cases} 1, & \text{if } x_i \neq 0 \text{ for all } i. \\ 0, & \text{else.} \end{cases}$$

Since there are $(p-1)^{|A|}$ points $x \in D_A$ comprised only of nonzero entries, the claim follows.

For (ii), we choose an arbitrary $n \in \mathbb{N}$ and proceed by induction on $|A|$. To see why the base case $|A| = 1$ holds, write $A = \{k\}$. For any $\alpha \in \mathcal{M}_n$ that is not an element of $[\top]_A$, we know $\alpha(k) \neq p-1$. Since $A = \{k\}$, we can simplify the sum

$$\sum_{x \in D_A} \prod_{i \in A} x_i^{\alpha(i)} = \sum_{r \in \mathbb{F}} r^{\alpha(k)}.$$

If $\alpha(k) = 0$, this is equal to zero. Otherwise, if $\alpha(k) \in \{1, \dots, p-2\}$, we use the definition of the discrete logarithm to write $r = \pi_p^{\lambda(r)}$ for any $r \in \mathbb{F}^\times$, and apply Lemma 4.4 to conclude the statement.

For the inductive case, let A be a subset of $[n]$ of size $|A| > 1$. Let $\alpha \in \mathcal{M}_n \setminus [\top]_A$. We consider the term $\sum_{x \in D_A} \prod_{i \in A} x_i^{\alpha(i)}$. Set $A = A' \cup \{k\}$ with $|A'| = |A| - 1$. We can rewrite the term in the following manner:

$$\sum_{x \in D_A} \prod_{i \in A} x_i^{\alpha(i)} = \sum_{x' \in D_{A'}} \sum_{r \in \mathbb{F}} \prod_{i \in A'} (x_i')^{\alpha(i)} \cdot r^{\alpha(k)} = \sum_{r \in \mathbb{F}} r^{\alpha(k)} \cdot \sum_{x' \in D_{A'}} \prod_{i \in A'} (x_i')^{\alpha(i)}.$$

If $\alpha(i) \neq p-1$ for some $i \in A'$, the second sum is equal to zero by induction hypothesis. Otherwise, since $\alpha \notin [\top]_A$, we have $\alpha(k) \neq p-1$, so by the base case $|A| = 1$, the first sum is equal to zero. This concludes the proof. □

Self-Reduction

The motivation for the mod p trick is the following: since the counting polynomial has high degree in general, we want to avoid having to compute an explicit representation of it in monomial basis. For this reason, we construct an approximation, whose structure allows for a simple self-reduction. Recall that the probabilistic counting polynomial $\widetilde{F}$ is the product

$$\widetilde{F}(X) = \prod_{i=1}^{\lceil \log_p \varepsilon^{-1} \rceil} \left(1 - (R_i(X))^{p-1}\right)$$

of independent probabilistic polynomials. Each of these R_i is a random linear combination

$$R_i(X) = \sum_{j=1}^{m} \rho_{ij} P_j(X)$$

[1] This abuse of notation is meant to signify that the indices of the formal variables in Φ are given by the elements of B.

of polynomials, where the ρ_{ij} are distributed uniformly on $\{0, \ldots, p-1\}$. By definition, the sum of a part $f|_A^{z\to B}$ is given by

$$I_{f|_A^{z\to B}} = \sum_{x \in D_A} f(x+z).$$

This sum can be reinterpreted as the sum over evaluations of a polynomial $f^{(z)}$ across its domain D_A, where $f^{(z)}$ is obtained by fixing the variables of f whose indices are contained in B to the values given by $z \in D_B$. Fixing variables in $\widetilde{F}$ can be achieved by fixing the variables in each linear combination $R_i(X)$. We write the polynomial obtained by fixing assignments according to z as $R_i^{(z)}(X_A)$, where X_A is shorthand for $(X_i \ ; \ i \in A)$, i.e., the unassigned variables. We can now write $\widetilde{F}^{(z)}(X_A)$ as

$$\widetilde{F}^{(z)}(X_A) = \prod_{i=1}^{\lceil \log_p \varepsilon^{-1} \rceil} \left(1 - \left(R_i^{(z)}(X_A)\right)^{p-1}\right).$$

By construction, the sum of $\widetilde{F}^{(z)}$ on all points of the domain D_A is equal to the number of solutions of the system of polynomial equations

$$R_1^{(z)}(X_A) = 0 \quad R_2^{(z)}(X_A) = 0 \quad \ldots \quad R_{\lceil \log_p \varepsilon^{-1} \rceil}^{(z)}(X_A) = 0. \tag{4.6}$$

This is an instance of our original problem in fewer variables. This is a straightforward generalisation of the self-reduction technique employed by Björklund et al. [6].

In summary, each $z \in D_B$ constitutes an instance of our original problem in $|A| = n - |B|$ variables by assigning the values given by z to the variables whose index is contained in B. For each $z \in D_B$, the return value of the recursive instance given by the system of polynomial equations (4.6) is equal to $I_{\widetilde{F}|_A^{z\to B}}$.

Back and Forth

For every recursive call on the polynomials obtained by fixing assignments according to some $z \in D_B$, we obtain the sum of the part $I_{\widetilde{F}|_A^{z\to B}}$. When $\widetilde{F}$ is written as $\sum_{\alpha \in \mathcal{M}_n} \widetilde{s}_\alpha X^\alpha$, the mod p trick (Corrolary 4.10) implies that this is equal to

$$I_{\widetilde{F}|_A^{z\to B}} = \sum_{\alpha \in [\top]_A} \widetilde{s}_\alpha \cdot \prod_{i \in B} z_i^{\alpha(i)} \cdot (p-1)^{|A|}. \tag{4.7}$$

The expression (4.7) looks awfully similar to the evaluation of a $|B|$-variate polynomial. We define a polynomial Φ whose variables are indexed by B and set the coefficients of its monomial representation $\Phi(X_B) = \sum_{\beta \in \mathcal{M}_B} t_\beta \cdot X_B^\beta$ according to

$$t_\beta := \widetilde{s}_\gamma \cdot (p-1)^{|A|},$$

$$\text{where } \gamma \in [\top]_A \text{ is given by } \gamma(i) = \begin{cases} \beta(i), & \text{if } i \in B \\ p-1, & \text{otherwise,} \end{cases}$$

it is evident that $\Phi(z) = I_{\widetilde{F}|_A^{z\to B}}$. Note that the degree of Φ is $\deg \Phi = \deg \widetilde{F} - (p-1) \cdot |A|$. This is the key to the recursive application of our algorithm. The recursive call on the instance obtained by assigning values according to some $z \in D_B$ returns $\Phi(z)$.

We make $s = \left\lceil \ln p \cdot n \cdot \frac{4(1-p^{-2})}{(p^{-2}-\frac{1}{2})^2} \right\rceil$ recursive calls (see p.31) and take the majority vote for each $z \in D_B$ that satisfies $\|z\|_1 \leq \deg \Phi$ to obtain a *truncated evaluation representation* of Φ. A single pass of the Injection Algorithm (Algorithm 3.6) returns the desired *full evaluation representation* of Φ. We can compute the sum of all evaluations of Φ to obtain the final result, i.e., the number of solutions of the system of polynomial equations

$$P_1(X) = 0 \quad P_2(X) = 0 \quad \ldots \quad P_m(X) = 0.$$

This concludes the description of the algorithm.

4.2 Running Time Analysis

We review the major steps of the algorithm. Let $\lambda = \lambda(d, p) \in (0, 1)$ denote a parameter which we are going to set later.

1. Draw $s = \left\lceil \ln p \cdot n \cdot \frac{4(1-p^{-2})}{(p^{-2}-\frac{1}{2})^2} \right\rceil = O(n)$ independent Razborov-Smolensky approximations (p.30) $f_1, \ldots, f_s$ of the counting polynomial (p.30).
2. Fix a partition $[n] = A \dot\cup B$ such that $|A| = \lfloor \lambda n \rfloor =: \ell$ holds (p.31).
3. Make a recursive call on all s approximations (p.35) and take the majority vote across approximations at every $z \in D_B$ which satisfies $\|z\|_1 \leq \deg \Phi$ (p.36).
4. Run the Injection Algorithm (Algorithm 3.6, p.24) over the result of all majority votes to obtain a full evaluation representation of Φ (p.36).
5. Sum over all evaluations of Φ to obtain the final result.

Let the constant $d > 1$ be the degree of the system and let $T(n, m) \in \mathbb{N}$ be the number of steps required to run the algorithm for a degree-d system of m polynomials in n variables on an RTM in the worst case.

We make s recursive calls for every $z \in D_B$ that satisfies $\|z\|_1 \leq \deg \Phi$. The degree of Φ can be upper-bounded by observing that the following bound holds:

$$\deg \Phi = \widetilde{F} - (p-1) \cdot \ell \leq (d-1) \cdot \ell \cdot (p-1) + 2d \cdot (p-1). \tag{4.8}$$

We set δ to the right-hand side of (4.8) and notice that there are $\kappa_n^{\leq \delta}$ such elements z satisfying the 1-norm bound. We need the following Lemma, which is taken from the proof of Lemma 3.3 in [19]:

4.11 Lemma ([19]). The following inequality holds for all $n, \Delta \in \mathbb{N}$:

$$\kappa_n^{\leq \Delta+1} \leq (n+1) \cdot \kappa_n^{\leq \Delta}.$$

In words: the number of points in $\mathbb{F}^n$ whose 1-norm is bounded by $\Delta + 1$ is at most $(n+1)$ times the number of points whose 1-norm is bounded by Δ. Intuitively this holds since for each point $x \in \mathbb{F}^n$ which satisfies $\|x\|_1 = \Delta$, one can obtain at most n distinct points x' satisfying $\|x'\|_1 = \Delta + 1$ by increasing one of the components of x by 1. We notice that we can set $\delta' := \delta - 2d(p-1) = (d-1)\ell(p-1)$ and apply Lemma 4.11 to write

$$\kappa_{n-\ell}^{\delta} \leq (n-\ell+1)^{2d(p-1)} \cdot \kappa_{n-\ell}^{\leq \delta'} = O\left(\kappa_{n-\ell}^{\leq \delta'} \cdot \mathrm{poly}(n)\right),$$

since d is a constant. This is a bound for the number of points for which we make a recursive call on an instance in ℓ variables (see p.36). As the polynomials given as input are represented by their monomial coefficients, such a representation needs to be built for each recursive call. First, the monomial coefficients of all polynomials given as inputs need to be summed over in order to build the representation of each Razborov-Smolensky approximation. Then, for each recursive call, the coefficients need to be looped over in order to assign values to a subset of the variables. The first operation takes

$$O\left(s \cdot (\ell+2) \cdot m \cdot \kappa_n^{\leq d}\right) = O\left(\kappa_n^{\leq d} \cdot \mathrm{poly}(n,m)\right)$$

steps since each of the s independent Razborov-Smolensky samples is made up of $\ell+2$ linear combinations of the m polynomials given as input. Since d is a constant, $\kappa_n^{\leq d}$ can be upper-bounded by $O(n^d \cdot (p-1)^d)$, hence the cost of the first operation is

$$O(\mathrm{poly}(n,m)).$$

The total cost of the second operation is

$$O\left(s \cdot (\ell+2) \cdot \kappa_n^{\leq d} \cdot \kappa_{n-\ell}^{\leq \delta}\right) = O\left(\kappa_{n-\ell}^{\leq \delta'} \cdot \mathrm{poly}(n)\right),$$

since each of the s Razborov-Smolensky approximation is made up of $\ell+2$ polynomials whose degree does not exceed d, and the value assignments need to be made for every point z in D_B whose 1-norm less than or equal to δ. Recalling that the running time of the Injection Algorithm (Algorithm 3.6) is bounded by $O(p^n \cdot n^2)$ for polynomials in n variables, we can give the following bound for $T(n,m)$:

$$T(n,m) = O^*\left(\kappa_{n-\ell}^{\leq \delta'} \cdot T(\ell, \ell+2) + \kappa_{n-\ell}^{\leq \delta'} + p^{n-\ell}\right). \tag{4.9}$$

The first summand accounts for the running time of the recursive calls, the second for the steps associated to building monomial representations and the third for the single pass of the Injection Algorithm on a polynomial in $n-\ell$ variables (see p.36). Recall that ℓ is defined as $\lfloor \lambda n \rfloor$. We omit the floor parentheses and write:

$$T(n,m) = O^*\left(\kappa_{(1-\lambda)n}^{\leq \delta'} \cdot T(\lambda n, \lambda n + 2) + p^{(1-\lambda)n}\right). \tag{4.10}$$

We set our reduction parameter λ such that $\delta' < \frac{1-\lambda}{2}n$, i.e., we require that $\lambda < \frac{1}{2(d-1)(p-1)+1}$ is satisfied. This allows us to use the following bound for $\kappa_{(1-\lambda)n}^{\leq \delta'}$:

$$\kappa_{(1-\lambda)n}^{\leq \delta'} \leq \frac{(1-\lambda)n}{2} \cdot \binom{(1-\lambda)n}{\delta'} \cdot (p-1)^{\delta'}. \tag{4.11}$$

Since $\frac{(1-\lambda)n}{\delta'} < \frac{1}{2}$ holds, we use the binary entropy function $H : [0,1] \to [0,1]$ given by

$$H(\rho) := -\rho \log_2(\rho) - (1-\rho)\log_2(1-\rho)$$

and use the fact that the inequality $\binom{n}{k} \leq 2^{H(k/n)n}$ holds for $n \in \mathbb{N}$ and $k \leq n/2$ [6]. This gives us a bound on the right-hand side of (4.11):

$$\begin{aligned}
\kappa_{(1-\lambda)n}^{\leq \delta'} &\leq \frac{(1-\lambda)n}{2} \cdot \binom{(1-\lambda)n}{\delta'} \cdot (p-1)^{\delta'} \\
&\leq \frac{(1-\lambda)n}{2} \cdot 2^{(1-\lambda)n \cdot H\left(\frac{\delta'}{(1-\lambda)n}\right)} \cdot (p-1)^{\delta'} \\
&= \frac{(1-\lambda)n}{2} \cdot p^{\log_p(2) \cdot (1-\lambda)n \cdot H\left(\frac{\delta'}{(1-\lambda)n}\right)} \cdot p^{\log_p(p-1) \cdot \delta'} \\
&= O^*\left(p^{n\left(H\left(\frac{(d-1)(p-1)\lambda}{(1-\lambda)n}\right) \cdot (1-\lambda) \cdot \log_p(2) + \lambda \cdot (d-1)(p-1) \cdot \log_p(p-1)\right)}\right).
\end{aligned} \tag{4.12}$$

We inserted $\delta' = (d-1)(p-1)\lambda$ in the last row. The same upper-bound can be found in [6] for the special case $p = 2$. Inserting (4.12) into (4.10), we get

$$T(n,m) = O^*\left(p^{n\left(H\left(\frac{(d-1)(p-1)\lambda}{(1-\lambda)n}\right)\cdot(1-\lambda)\cdot\log_p(2)+\lambda\cdot(d-1)(p-1)\cdot\log_p(p-1)\right)}\cdot T(\lambda n, \lambda n+2) + p^{(1-\lambda)n}\right). \tag{4.13}$$

Just like the authors of [6], we notice that we can define a parameter $\tau(k) \in \mathbb{R}_{>0}$ for $k = 0, \ldots, D$ for some constant $D = D(\lambda)$ such that

$$T(n,m) = O^*\left(p^{\tau(0)n}\right).$$

The constant D is our recursion depth, hence we set $\tau(D) := \lambda^D$ to account for the brute-force base case. For $k = 0, \ldots, D-1$, we set

$$\tau(k) := \lambda^k \cdot \max\left\{\begin{array}{c} H\left(\frac{(d-1)(p-1)\lambda}{(1-\lambda)n}\right)\cdot(1-\lambda)\cdot\log_p(2) + \lambda\cdot(d-1)(p-1)\cdot\log_p(p-1) + \tau(k+1), \\ 1-\lambda \end{array}\right\}.$$

The parameter $\tau(0)$ yields the exponent of p in our total running time. At each level of recursion, the running time is given by the term that dominates the sum (4.13). Our goal is to force $\tau(0) = 1-\lambda$. In other words: we want to set the parameter λ such that the total running time is equal to the time required by the Injection Algorithm. By observing the definition of $\tau(k)$, we notice that in order to obtain $\tau(0) = 1-\lambda$, it suffices to choose a λ such that

$$H\left(\frac{(d-1)(p-1)\lambda n}{(1-\lambda)n}\right)\cdot(1-\lambda)\cdot\log_p(2) + \lambda\cdot(d-1)(p-1)\cdot\log_p(p-1) + \lambda(1-\lambda) \leq 1-\lambda, \tag{4.14}$$

holds and set D to a large enough value such that

$$\tau(D-1) \leq 1-\lambda \tag{4.15}$$

holds. If both (4.14) and (4.15) hold, then $\tau(0)$ takes on the value $1-\lambda$ as we desire. We inspect both inequalities. We insert the definition of $\tau(D-1)$ into (4.15) and obtain

$$H\left(\frac{(d-1)(p-1)\lambda}{(1-\lambda)n}\right)\cdot(1-\lambda)\cdot\log_p(2) + \lambda\cdot(d-1)(p-1)\cdot\log_p(p-1) + \lambda^D \leq 1-\lambda.$$

This inequality is equivalent to

$$H\left(\frac{(d-1)(p-1)\lambda}{(1-\lambda)n}\right) \leq \left(1-\frac{\lambda^D}{1-\lambda}\right)\log_2(p) - \frac{(d-1)(p-1)\lambda}{1-\lambda}\log_2(p-1). \tag{4.16}$$

We now rearrange (4.20) and obtain

$$H\left(\frac{(d-1)(p-1)\lambda}{1-\lambda}\right) \leq (1-\lambda)\log_2(p) - \frac{(d-1)(p-1)\lambda}{1-\lambda}\log_2(p-1). \tag{4.17}$$

Comparing (4.16) and (4.17), we notice the following: if the recursion depth D is set to a sufficiently large value such that

$$\frac{\lambda^D}{1-\lambda} < \lambda$$

holds, then the satisfaction of (4.17) implies the desired bound $\tau(0) = 1 - \lambda$.

In the following section we aim to prove a simple running time bound. To obtain a simple bound, we choose some $a > 3$, set $\lambda := \frac{1}{a(d-1)(p-1)}$ and observe:

$$H\left(\frac{(d-1)(p-1)\lambda}{1-\lambda}\right) = H\left(\frac{(d-1)(p-1)}{a(d-1)(p-1)-1}\right) \in \left(H\left(\frac{1}{a}\right), H\left(\frac{1}{a-1}\right)\right], \tag{4.18}$$

so $H\left(\frac{(d-1)(p-1)}{a(d-1)(p-1)-1}\right) \leq H\left(\frac{1}{a-1}\right)$ holds for all choices of d and p. Inspecting the right-hand side of (4.17), we get

$$(1-\lambda)\log_2(p) = \frac{a(d-1)(p-1)-1}{a(d-1)(p-1)}\log_2(p) \in \left[\frac{a-1}{a}\log_2(p), \log_2(p)\right), \text{ and}$$
$$\frac{(d-1)(p-1)\lambda}{1-\lambda}\log_2(p-1) = \frac{(d-1)(p-1)}{a(d-1)(p-1)-1}\log_2(p-1) \in \left(\frac{1}{a}\log_2(p-1), \frac{1}{a-1}\log_2(p-1)\right],$$

so it follows that

$$(1-\lambda)\log_2(p) - \frac{(d-1)(p-1)\lambda}{1-\lambda}\log_2(p-1) \tag{4.19}$$
$$\in \left[\frac{a-1}{a}\log_2(p) - \frac{1}{a-1}\log_2(p-1), \log_2(p) - \frac{1}{a}\log_2(p-1)\right).$$

A Simple Bound

We set out to prove the following running time bound:

4.12 Theorem. The running time of the algorithm described in section 4.1 can be upper-bounded by

$$O^*\left(p^{(1-\lambda)n}\right),$$

where $\lambda = \frac{1}{5.1(d-1)(p-1)}$ and d is the degree of the system.

To aid our reasoning, we state two basic observations concerning the lower bound of the expression given in (4.19). We will use these observations to find a constant $a > 3$ such that $H(1/(a-1)) < (a-1)/a\log_2(p) - 1/(a-1)\log_2(p-1)$ holds for all $p \geq 2$. As we will notice further on, this constant constitutes a loose bound as p grows large, though we will be content with a loose bound for now. We start with the following elementary analytical property of the aforementioned lower bound, when it is considered as a real function.

4.13 Lemma. For all $a \geq 4$, the function

$$g_1^{(a)} : \mathbb{R} \to \mathbb{R}$$
$$p \mapsto \frac{a-1}{a}\log_2(p) - \frac{1}{a-1}\log_2(p-1)$$

is strictly increasing as p ranges over the interval $[2, \infty)$.

Proof. We show that the derivative of g_a is strictly positive on $[2, \infty)$ for all $a \geq 4$. The inequality which needs to be satisfied by all $p \in [2, \infty)$ is given by

$$\frac{d}{dp}g_1^{(a)}(p) = \frac{(a-1)^2(p-1)\ln(2) - ap\ln(2)}{a(a-1)p\ln(2)(p-1)\ln(2)} > 0$$
$$\Leftrightarrow (a-1)^2(p-1) - ap > 0.$$

Since the derivative of the left-hand-side function above itself is strictly positive for $a \geq 3$, it suffices to find an $a \geq 3$ that satisfies the inequality for $p = 2$. The inequality can then be written as

$$(a-1)^2 - 2a > 0,$$

which is satisfied by every $a \geq 4$. □

Next, we show that the aforementioned lower bound, considered as a function of p, is also strictly increasing.

4.14 Lemma. For all $p \geq 2$, the function

$$\begin{aligned} g_2^{(p)} : \mathbb{R} &\to \mathbb{R} \\ a &\mapsto \frac{a-1}{a}\log_2(p) - \frac{1}{a-1}\log_2(p-1) \end{aligned}$$

is strictly increasing as a ranges over the interval $[2, \infty)$.

Proof. For some fixed $p \geq 2$, its derivative is given by

$$\frac{d}{da} g_2^{(p)}(a) = \frac{1}{a^2}\log_2(p) + \frac{1}{(a-1)^2}\log_2(p-1),$$

which is evidently strictly positive for $a \geq 2$. □

Putting together Lemma 4.13 and Lemma 4.14, we can obtain a simple bound on the running time by finding a sufficiently small $a \geq 4$ which satisfies the inequality

$$H\left(\frac{1}{a-1}\right) < g_1^{(a)}(2) = \frac{a-1}{a}.$$

Numerical exploration yields $a = 5.1$ as a solution. By Lemma 4.13, this bound is satisfied for all $p \geq 2$, as $g_1^{(5.1)}(p)$ is strictly increasing. This gives us an upper bound of $O^*\left(p^{(1-\lambda)n}\right)$ for the running time of the algorithm described in section 4.1, where $\lambda = \frac{1}{5.1(d-1)(p-1)}$. Note that the bound is only tight for $p = 2$. We inspect the inequality

$$H\left(\frac{1}{a-1}\right) < \frac{a-1}{a}\log_2(p) - \frac{1}{a-1}\log_2(p-1). \tag{4.20}$$

Fix an $a \geq 2$. Since the left-hand side of (4.20) is constant, and the right-hand side is strictly increasing as a function of p (Lemma 4.13), we notice that setting a to a constant value is suboptimal. Even though we aim to improve the running time bound, this is beyond the scope of this thesis.

5 Conclusions

5.1 What We Did

In this thesis we generalised an algorithmic idea that rests on some properties of the two-element field $\mathbb{F}_2$ to $\mathbb{F}_p$. The algorithm we transferred to the $\mathbb{F}_p$ setting was based on a reduction from decision to counting and a recursive application [6]. It was noticeable that many of the ideas which were employed over $\mathbb{F}_2$ were merely special cases of statements over $\mathbb{F}_p$. We obtained an improvement over the running time of [19] for small primes, though the bound reported by Lokshtanov et al. is presumably not tight.

5.2 What Remains To Be Done

While the main objective of the thesis has been accomplished, not all work is done yet.

Tight Running Time Bounds

Out of the things that remain to be done, finding tight bounds for the running time is probably the most pressing. The simple bound given in the running time analysis (Section 4.2) is not tight for any prime $p > 2$, hence it ignores a whole dimension in the parameter space. Finding tighter bounds should only be a matter of deeper analysis of the formulas that have already been identified in this thesis. On a different note, large portions of the running time analysis can maybe be overhauled for the sake of better presentation.

All Fields of Prime Characteristic

The generalisation presented in this thesis is only formulated for a tiny fraction of finite fields, as we only considered finite fields of *prime* order. Those fields are friendly to work with, as they are given by the quotient rings $\mathbb{Z}/p\mathbb{Z}$. The Fundamental Theorem of Finite Fields says that there exists a finite field of order p^k for every prime p and every natural number k [2]. It might be interesting to see whether, with a little care, the algorithm can be formulated over $\mathbb{F}_{p^k}$ as well.

Hardness

In the context of conditional lower bounds based on the exponential time hypothesis (see [18]), it might be interesting to formulate fine-grained reductions to systems of polynomial equations over $\mathbb{F}_p$. This would contextualise further progress to be made in the design of algorithms for systems of polynomial equations and could motivate the study of other problems in algebra through an algorithmic lens.

Further Ramblings

Probably one of the oldest realisations about mathematics (and subsequently about computer science) is that generalisation leads to improvements in understanding. Even though the two-element field is a natural algebraic setting for many problems in computer science, it might be fun and interesting to try to generalise more algorithmic ideas which rest on some properties of the finite field of order 2.

Bibliography

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[2] Michael Artin. *Algebra*. Basel [u.a.], 1998. URL: http://scans.hebis.de/HEBCGI/show.pl?05684971_toc.pdf.

[3] Michael Atiyah. *Introduction to Commutative Algebra*. Westview Press, 1994.

[4] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Pierre-Jean Spaenlehauer. "On the complexity of solving quadratic boolean systems". In: *Journal of Complexity* 29.1 (2013), pp. 53–75.

[5] Rajendra Bhatia. *Matrix Analysis*. New York [u.a.], 1997. URL: http://scans.hebis.de/HEBCGI/show.pl?05147375_toc.pdf.

[6] Andreas Björklund, Petteri Kaski, and Ryan Williams. "Solving Systems of Polynomial Equations over GF(2) by a Parity-Counting Self-Reduction". In: *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2019.

[7] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Vol. 4. 8. Springer, 2015.

[8] Jintai Ding and Bo-Yin Yang. "Multivariate Public Key Cryptography". In: *Post-Quantum Cryptography*. Springer, 2009, pp. 193–241.

[9] Itai Dinur. "Improved Algorithms for Solving Polynomial Systems over GF (2) by Multiple Parity-Counting". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2021, pp. 2550–2564.

[10] Steffen Eger. "Restricted Weighted Integer Compositions and Extended Binomial Coefficients". In: *J. Integer Seq* 16.13.1 (2013), p. 3.

[11] Aviezri S Fraenkel and Yaacov Yesha. "Complexity of Problems in Games, Graphs and Algebraic Equations". In: *Discrete Applied Mathematics* 1.1-2 (1979), pp. 15–30.

[12] Michael L Fredman and Dan E Willard. "Blasting Through the Information Theoretic Barrier with Fusion Trees". In: *Proceedings of the twenty-second annual ACM symposium on Theory of Computing*. 1990, pp. 1–7.

[13] Silvia Heubach and Toufik Mansour. "Compositions of n with Parts in a Set". In: *Congressus Numerantium* 168 (2004), p. 127.

[14] Neil Immerman. *Descriptive Complexity*. New York [u.a.], 1999. URL: http://scans.hebis.de/HEBCGI/show.pl?06260607_toc.pdf.

[15] Zahra Jafargholi, Hamidreza Jahanjou, Eric Miles, Jaideep Ramachandran, and Emanuele Viola. *From RAM to SAT (unpublished project paper)*. 2012. URL: http://www.ccs.neu.edu/home/viola/papers/ram2sat.pdf.

[16] Gašper Jaklič, Vito Vitrih, and Emil Žagar. "Closed form formula for the number of restricted compositions". In: *Bulletin of the Australian Mathematical Society* 81.2 (2010), pp. 289–297.

[17] Allen Klinger. "The Vandermonde Matrix". In: *The American Mathematical Monthly* 74.5 (1967), pp. 571–574.

[18] Daniel Lokshtanov, Dániel Marx, Saket Saurabh, et al. "Lower Bounds Based on the Exponential Time Hypothesis". In: *Bulletin of EATCS* 3.105 (2013).

[19] Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, Ryan Williams, and Huacheng Yu. "Beating Brute Force for Systems of Polynomial Equations over Finite Fields". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2017, pp. 2190–2202.

[20] Kurt Mehlhorn. *Algorithms and Data Structures: The Basic Toolbox*. Berlin [u.a.], 2008. URL: http://d-nb.info/987221353/04.

[21] Dieter van Melkebeek. *CS 810: Complexity Theory Lecture Notes; Introduction*. 2007. URL: http://pages.cs.wisc.edu/~dieter/Courses/2007s-CS810/Scribes/PS/lectures.pdf.

[22] Michael Mitzenmacher. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge [u.a.], 2005. URL: http://scans.hebis.de/HEBCGI/show.pl?12671652_toc.pdf.

[23] Alexander A. Razborov. "Lower Bounds on the Size of Bounded Depth Circuits over a Complete Basis with Logical Addition". In: *Mathematical Notes of the Academy of Sciences of the USSR* 41.4 (1987), pp. 333–338.

[24] Roman Smolensky. "Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity". In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. 1987, pp. 77–82.

[25] Jakob Stix. "Grundlagen der Algebra". In: *Vorlesungsskript, Goethe-Universität Frankfurt/Main* (2019).

[26] Jakob Stix, Quentin Gendron, and Kolja Hept. "Kommutative Algebra". In: *Vorlesungsskript, Goethe-Universität Frankfurt/Main* (2015).

[27] Leslie G. Valiant and Vijay V. Vazirani. "NP is as Easy as Detecting Unique Solutions". In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. 1985, pp. 458–463.

[28] Ryan Williams. "Guest column: A Casual Tour Around a Circuit Complexity Bound". In: *ACM SIGACT News* 42.3 (2011), pp. 54–76.

[29] Bernard Ycart. "A Case of Mathematical Eponymy: The Vandermonde Determinant". In: *arXiv preprint arXiv:1204.4716* (2012).

A Appendix

A.1 A Visualisation of the Multiplication Table of $(\mathbb{F}_7)^\times$

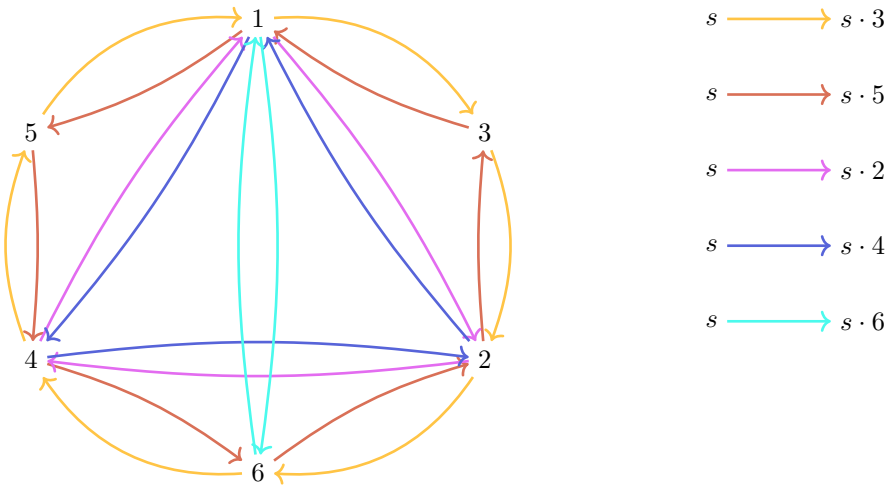

Figure A.1: A visualisation of the multiplication table of $(\mathbb{F}_7)^\times$. An edge symbolises multiplication of the value at the source node with the element associated with the color of the edge, resulting in the value at the target node. Note that both 3 and 5 are primitive elements.

A.2 Inverse of the Univariate Vandermonde Matrix

For all primes p, the Vandermonde matrix $V := \mathsf{Van}(0, \dots, p-1) \in \mathbb{F}_p^{p \times p}$ is invertible. We give an explicit inverse V^{-1} and prove that $VV^{-1} = \mathrm{id}$.

A.1 Proposition (Inverse of Vandermonde matrix)**.** Let $p \in \mathbb{N}$ be a prime. The inverse of the Vandermonde matrix

$$V := \mathsf{Van}(0, \dots, p-1) \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots & 2^{p-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & p-1 & (p-1)^2 & \cdots & (p-1)^{p-1} \end{pmatrix}$$

in $\mathbb{F}_p$ is the matrix

$$V^{-1} := \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & v_{1,1} & v_{1,2} & \cdots & v_{1,p-1} \\ 0 & v_{2,1} & v_{2,2} & \cdots & v_{2,p-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & v_{p-2,1} & v_{p-2,2} & \cdots & v_{p-2,p-1} \\ p-1 & v_{p-1,1} & v_{p-1,2} & \cdots & v_{p-1,p-1} \end{pmatrix},$$

where $v_{i,j} = (p-1) \cdot j^{-i} \in \mathbb{F}_p$.

Proof. In order to show that the matrix product

$$VV^{-1} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots & 2^{p-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & p-2 & (p-2)^2 & \cdots & (p-2)^{p-1} \\ 1 & p-1 & (p-1)^2 & \cdots & (p-1)^{p-1} \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & v_{1,1} & v_{1,2} & \cdots & v_{1,p-1} \\ 0 & v_{2,1} & v_{2,2} & \cdots & v_{2,p-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & v_{p-2,1} & v_{p-2,2} & \cdots & v_{p-2,p-1} \\ p-1 & v_{p-1,1} & v_{p-1,2} & \cdots & v_{p-1,p-1} \end{pmatrix}$$

is equal to id, it suffices to show that the product of the minors

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 2 & 2^2 & \cdots & 2^{p-1} \\ \vdots & \vdots & \ddots & \vdots \\ p-1 & (p-1)^2 & \cdots & (p-1)^{p-1} \end{pmatrix} \cdot \begin{pmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,p-1} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{p-1,1} & v_{p-1,2} & \cdots & v_{p-1,p-1} \end{pmatrix} =: W$$

is equal to id. The remaining entries of the product VV^{-1} can be verified easily by hand. When W is written as $(w_{i,j})$, the statement $W = \mathrm{id}$ is equivalent to the statement that

$$w_{i,j} = \sum_{k=1}^{p-1} i^k \cdot w_{k,j} = (p-1) \cdot \sum_{k=1}^{p-1} i^k \cdot j^{-k} = (p-1) \cdot \sum_{k=1}^{p-1} \left(\frac{i}{j}\right)^k = \delta_{i,j},$$

where $\delta_{i,j}$ denotes the Kronecker delta. We write $\frac{i}{j}$ to denote the element $i \cdot j^{-1}$ of $(\mathbb{F}_p)^\times$. If i is equal to j, the fraction $\frac{i}{j}$ is equal to 1 and the expression is equal to $(p-1)(p-1) = 1$. If i is not equal to j, the fraction $\frac{i}{j}$ is not equal to 1 and we can apply the formula for geometric sums. This yields

$$(p-1) \cdot \sum_{k=1}^{p-1} \left(\frac{i}{j}\right)^k = (p-1) \cdot \left(\frac{\left(\frac{i}{j}\right)^p - 1}{\frac{i}{j} - 1} - 1 \right) = (p-1) \cdot (1-1) = 0.$$

This concludes the proof. □