Journal of Complexity 73 (2022) 101681

Contents lists available at ScienceDirect

Journal of Complexity

www.elsevier.com/locate/jco

Computing zeta functions of large polynomial systems over finite fields ☆

Qi Cheng [a], J. Maurice Rojas [b,*], Daqing Wan [c]

[a] University of Oklahoma, School of Computer Science, Norman, OK 73019, United States of America
[b] Texas A&M University, Department of Mathematics, College Station, TX 77843-3368, United States of America
[c] University of California, Irvine, Department of Mathematics, Irvine, CA 92697-3875, United States of America

ARTICLE INFO

Article history:
Received 22 August 2021
Received in revised form 7 April 2022
Accepted 19 May 2022
Available online 1 June 2022

MSC:
11Y16
11G25
14G10

Keywords:
Counting solutions
Finite fields
Program verification

ABSTRACT

We improve the algorithms of Lauder-Wan [11] and Harvey [8] to compute the zeta function of a system of m polynomial equations in n variables, over the q element finite field $\mathbb{F}_q$, for large m. The dependence on m in the original algorithms was exponential in m. Our main result is a reduction of the dependence on m from exponential to polynomial. As an application, we speed up a doubly exponential algorithm from a recent software verification paper [3] (on universal equivalence of programs over finite fields) to singly exponential time. One key new ingredient is an effective, finite field version of the classical Kronecker theorem which (set-theoretically) reduces the number of defining equations for a polynomial system over $\mathbb{F}_q$ when q is suitably large.

1. Introduction

Let $\mathbb{F}_q$ be the finite field of cardinality q with characteristic p. Let F be a polynomial system with m equations and n variables over $\mathbb{F}_q$:

$$F(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)),$$

☆ Communicated by K.K. Meer.

* Corresponding author.
E-mail addresses: qcheng@ou.edu (Q. Cheng), jmauricerojas@gmail.com (J. Maurice Rojas), dwan@math.uci.edu (D. Wan).

https://doi.org/10.1016/j.jco.2022.101681
0885-064X/

where each $f_i \in \mathbb{F}_q[x_1, \ldots, x_n]$ has total degree at most d. Note that the total number of digits needed to write down the monomial term expansions in such a system is $O\left(m\binom{d+n}{n}\log q\right)$. So it is natural to use $m\binom{d+n}{n}\log q$ as a measure of *input size* for F when discussing algorithmic efficiency.

One could refine the input size further — to take sparsity into account — by replacing the $m\binom{d+n}{n}$ factor with a quantity closer to the number of monomial terms of F. (See, e.g., [7,4,1] where various **NP**-hardness and #**P**-hardness results are proved relative to this potentially smaller measure.) However, for applications like software verification, there is no reason to expect that the F one encounters will have few monomial terms. So we will stick with $m\binom{d+n}{n}\log q$ as our measure of size. For our purposes here, and for reasons to be made clear shortly, we will call the polynomial system F *large* if the number of equations m is at least $n+2$.

A basic algorithmic problem in number theory is to compute the number of solutions, $N_q(F)$, of the polynomial system $F = (0, \ldots, 0)$ over $\mathbb{F}_q$. More precisely, we let $N_q(F)$ denote the cardinality of $\left\{ (x_1, \ldots, x_n) \in \mathbb{F}_q^n \;\middle|\; F(x_1, \ldots, x_n) = (0, \ldots, 0) \right\}$. The special case $(m, n) = (1, 2)$ already plays a huge role in cryptography, since curves with a specified number of points are crucial to the design of many cryptosystems (see, e.g., [6]).

An even deeper problem is to consider all extension fields of $\mathbb{F}_q$ at once and compute the full sequence $N_q(F), N_{q^2}(F), \ldots, N_{q^k}(F), \ldots$ or, equivalently, the generating zeta function

$$Z(F, T) = \exp\left(\sum_{k=1}^{\infty} \frac{N_{q^k}(F)}{k} T^k\right).$$

Understanding this generating function occupied a good portion of 20th century algebraic and arithmetic geometry. Interestingly, this generating function has found a recent application to software engineering, specifically, in program equivalence [3]. (We clarify this in the next section.) It is not at all obvious from the definition that this zeta function is effectively computable, so let us briefly recall how it actually is.

A deep and celebrated theorem of Dwork from 1960 says that the zeta function is a rational function in T. A theorem of Bombieri [5] from 1988 says that the total degree of the zeta function is effectively boundable. It then follows, from basic manipulation of power series, that the zeta function is effectively computable, although practical efficiency is far more subtle: See [14] for a survey on algorithms for computing zeta functions. A general deterministic algorithm to compute $Z(F, T)$ was constructed in Lauder-Wan [11] with running time

$$2^m (p m^n d^n \log q)^{O(n)}.$$

One should note that the case $d = 1$ is easily handled by linear algebra, leading to a complexity bound of $O\left((\max^3\{m, n\}\log^2 q\right)$ via any reasonable implementation of Gaussian elimination and finite field arithmetic, without invoking sub-cubic matrix multiplication or Fast Fourier Transform multiplication in $\mathbb{F}_q$.

For small characteristic p, the general algorithm from [11] remains the best so far. However, for large characteristic p, the dependence on p has been improved by Harvey [8], who constructed an algorithm with running time

$$2^m p (m^n d^n \log q)^{O(n)},$$

for $d \geq 2$. (There is also a variant in [8] with time complexity linear in $\sqrt{p}$ instead, but at the expense of increasing the space complexity to roughly the same order as the time complexity.) The algorithms from [11] and [8] are, however, fully exponential in m, even for fixed n.

To improve the dependence on m, we briefly explain how the exponential factor 2^m arises in the algorithms of [11] and [8]: Both algorithms, in the case $m = 1$ (the hypersurface case), are obtained via p-adic trace formulas (meaning linear algebra with large matrices over the polynomial ring $(\mathbb{Z}/p^\lambda\mathbb{Z})[t]$, arising after some cohomological calculations). The case $m > 1$ is then reduced to the case $m = 1$ via an inclusion-exclusion trick [14] to compute the zeta function for each of

the 2^m hypersurfaces defined by $f_S = \prod_{i\in S} f_i$, where S runs through all subsets of $\{1, \ldots, m\}$ and $\deg(f_S) \leq |S|d \leq md$.

In this paper, we improve the Lauder-Wan algorithm and the Harvey algorithm by employing an additional algebraic trick to reduce the exponential factor from 2^m to m. A key new idea is to prove an effective, finite field version of Kronecker's theorem that enables us to reduce the number of defining equations, m, to $\min\{m, n+1\}$ when q is sufficiently large: See Section 3 below.

Our main result is the following:

Theorem 1.1. *There is an explicit deterministic algorithm which computes the zeta function $Z(F, T)$ of the system F over $\mathbb{F}_q$ (with m equations, n variables, and total degree at most d for each equation) in time $mp(n^n d^n \log q)^{O(n)}$.*

We prove Theorem 1.1 in Section 4.

We will see in the next section how our theorem enables us to speed up a *doubly* exponential time algorithm (from [3]) for program equivalence to *singly* exponential time. In particular, we will now briefly review some of the background on programs over finite fields.

2. Programs, their equivalence, and zeta functions

2.1. Background on program equivalence

A basic and difficult problem from the theory of programming languages is determining when two programs always yield the same output, *without* trying all possible inputs. This problem — a special case of *program equivalence* — also has an obvious parallel in cryptography: A fundamental problem there is to decide whether a putative key for an unknown stream cipher (that one has spent much time deducing) is correct or not, without trying all possible inputs. In full generality, program equivalence is known to be undecidable in the classical Turing model of computation. However, program equivalence (and *formal verification*, in greater generality [10]) remains an important problem in software engineering and cryptography. It is then natural to ask these questions in a more limited setting.

For instance, Barthe, Jacomme, and Kremer (in [3]) describe a programming language which enables a broad family of calculations (and verifications thereof) involving polynomials over finite fields. They proved that program equivalence in their setting is decidable, and gave an algorithm with doubly exponential complexity. We now briefly review their terminology (from [3, Sec. 2.2]), and explain how their algorithm requires a non-trivial use of zeta functions.

To be more precise, in their restricted setting, a *program* is a sequence of logical/polynomial expressions over a finite field. To define this rigorously, one first fixes a set I of *input* variables and a set R of *random* variables. Then all possible expressions making up a program can be defined recursively (building up from (1) and (2) below) as follows:

(1) a polynomial $P \in \mathbb{F}_q[I, R]$;
(2) the failure statement $\perp$;
(3) an "`if`" statement of the following form:

    `if` b `then` e_1 `else` e_2

    where e_1 and e_2 are expressions, and b is a propositional logic formula, whose atoms are of the form $Q = 0$ for some $Q \in \mathbb{F}_q[I, R]$.

Remark 2.1. Programs in [3] are written using semi-colons as delimiters, similar to some real-world program languages such as `C` or `Java`. ◇

The *size* of a program is defined to be the number of characters in a program. The presence of random variables enables our programs to use randomization, and give answers with a certain probability of failure. We denote the set of all such programs by $\mathcal{P}_q(I, R)$. Polynomials in a program are

represented by arithmetic formulas, so the degree of any polynomial in the program is bounded from above by the size of the program. Note that programs in this core language do not have loops. If a program has neither "`if`" statements nor failure statements then we call the program an *arithmetic program*. The set of all arithmetic programs is denoted by $\bar{\mathcal{P}}_q(I,R)$.

The *number of expressions at the top level* of a program $\mathbb{P}$ — denoted by $|\mathbb{P}|$ — is simply the length of the sequence defining $\mathbb{P}$. (In a real-world programming language, the "top level" of a program simply means one ignores subroutines and, e.g., statements *inside* of an "if" statement.) Note also that since our programs can use random variables, our programs thus send input values in $\mathbb{F}_{q^k}^{|I|}$ to a probability distribution over $\mathbb{F}_{q^k}^{|\mathbb{P}|}$, for any positive integer k. Here we assume that for every input assignment, the program does not fail (i.e., there is no evaluation of $\perp$ that halts the program) for at least one random assignment. A program can thus be viewed as a map from inputs to real-valued functions: $\mathbb{F}_{q^k}^{|I|} \to (\mathbb{F}_{q^k}^{|\mathbb{P}|} \to [0,1])$. The following example reveals how understanding the semantics of a program (i.e., what the program does) in fact requires counting solutions of polynomial systems over finite fields.

Example 2.2. Fixing $I=\emptyset$ and $R=\{x\}$, the program

$$x*x;\ x*x*x$$

outputs the square, and the cube, of a uniformly random element of $\mathbb{F}_q$.[1] Let $N(\alpha,\beta)$ denote the number of solutions in $\mathbb{F}_{q^k}$ of $(x^2,x^3)=(\alpha,\beta)$. The output of the program thus exhibits a random variable on $\mathbb{F}_{q^k}^2$, with probability mass function taking the value $N(\alpha,\beta)/q^k$ at (α,β). $\diamond$

Example 2.3. Let $I=\{x\}$ and $R=\{y,z\}$. The following program $\mathbb{P}_1$ is in $\mathcal{P}(I,R)$:

$$\texttt{if } \neg(x=0) \texttt{ then } y+1 \texttt{ else } y+2;\ \ z*z$$

The program $\mathbb{P}_1$ yields the probability distribution on $\mathbb{F}_{q^k}^2$ corresponding to the first coordinate being uniformly random in $\mathbb{F}_{q^k}$ and the second coordinate a uniformly random square in $\mathbb{F}_{q^k}$. $\diamond$

To calculate the distribution, the sample space consists of the assignments to random variables so that the program does not fail. Recall that we assume that for every input assignment, the program does not fail for at least one random assignment. For example, the following program ($I=\{x\}$ and $R=\{y\}$) computes the inverse of x with probability 1:

$$\texttt{if } x=0 \texttt{ then } 0 \texttt{ else } \texttt{ if } x*y=1 \texttt{ then } y \texttt{ else } \perp$$

Given two programs, we would like to check whether they produce the same distribution for any input. More generally, let $\mathbb{P}_1,\mathbb{Q}_1$ be programs and $\mathbb{P}_2,\mathbb{Q}_2$ be arithmetic programs. We write $\mathbb{P}_1|\mathbb{P}_2 \approx \mathbb{Q}_1|\mathbb{Q}_2$ if, taking any input c under the condition that $\mathbb{P}_2=\vec{0}$, $\mathbb{P}_1$ outputs the same distribution as $\mathbb{Q}_1$ taking c as input under the condition $\mathbb{Q}_2=\vec{0}$. To calculate the distribution, we only need to consider the random values such that none of $\mathbb{P}_1$ and $\mathbb{Q}_1$ output $\perp$.

Remark 2.4. Observe that the set of inputs yielding a fixed sequence of outputs is nothing more than a constructible set over $\mathbb{F}_{q^k}$, i.e., a boolean combination of algebraic sets over $\mathbb{F}_{q^k}$. In particular, the set of inputs making two programs *differ* is also a constructible set over a finite field. $\diamond$

The question of equivalence can be raised for a fixed k, or for all positive integers k. The latter case is called *universal equivalence*, which is most relevant to our discussion here. For example, let $\mathbb{Q}_1$ be the program defined by:

[1] The value of x is fixed once the first instance has taken place. Note also that we use $x*x$ in place of x^2, since polynomials are represented by arithmetic formulas and thus arbitrary exponentiation requires a varying number of gates.

$$y;\ \ 7*(z+1)*(z+1).$$

If 7 is a nonzero square in $\mathbb{F}_q$ then the program $\mathbb{P}_1$ (from Example 2.3) is universally equivalent to $\mathbb{Q}_1$, i.e., $\mathbb{P}_1|0 \approx \mathbb{Q}_1|0$. Otherwise, $\mathbb{P}_1|0$ and $\mathbb{Q}_1|0$ are not equivalent over $\mathbb{F}_q$, and hence not universally equivalent.

2.2. An improved reduction from general to arithmetic programs

Note that in greater generality, checking universal equivalence means checking if a *sequence* of constructible sets consists solely of empty sets (per Remark 2.4 above). As observed in [3], this can be done by a single zeta function computation. This is, in essence, how [3] proved that universal equivalence for arithmetic programs can be done in singly exponential time. For universal equivalence of conditional programs, the same ideas apply, but [3] proved a doubly exponential complexity upper bound. More precisely, for general programs $\mathbb{P}_1, \mathbb{Q}_1$ and arithmetic programs $\mathbb{P}_2, \mathbb{Q}_2$, they defined a reduction, to obtain four arithmetic programs $\mathbb{P}_1', \mathbb{P}_2', \mathbb{Q}_1'$ and $\mathbb{Q}_2'$ so that

$$\mathbb{P}_1|\mathbb{P}_2 \approx \mathbb{Q}_1|\mathbb{Q}_2 \textbf{ if and only if } \mathbb{P}_1'|\mathbb{P}_2' \approx \mathbb{Q}_1'|\mathbb{Q}_2'.$$

One can reduce *general* program equivalence to deciding $\mathbb{P}_1'|\mathbb{P}_2' \approx \mathbb{Q}_1'|\mathbb{Q}_2'$, where $\mathbb{P}_1', \mathbb{P}_2', \mathbb{Q}_1'$ and $\mathbb{Q}_2'$ are each *arithmetic* programs. More precisely, if ℓ is the input size of the original programs (i.e., the sum of the sizes of $\mathbb{P}_1, \mathbb{P}_2, \mathbb{Q}_1$ and $\mathbb{Q}_2$), then we can build the new arithmetic programs $\mathbb{P}_i'$ and $\mathbb{Q}_i'$ so that they have size $2^{O(\ell)}$, consist of $2^{O(\ell)}$ many polynomials, but involve just $\ell^{O(1)}$ many variables. Furthermore, the total degree of each polynomial in the new programs is at most $2^{O(\ell)}$.

We outline this reduction below, but first let us state an immediate consequence of applying our improved zeta function algorithm to this reduction: We improve the main complexity bound of [3] from doubly exponential to singly exponential.

Theorem 2.5. *Universal equivalence of programs can be checked in time singly exponential in the size.* ■

Since our main focus is speeding up zeta function computation, we will now briefly outline, through some representative examples, how to reduce general programs to arithmetic programs. See [3] for the full, formal treatment of their original reduction.

First, it is clear that we need to be able to remove failure statements ($\perp$) and "`if`" statements from $\mathbb{P}_1$ and $\mathbb{Q}_1$. We may assume that there is at most one occurrence of the failure statement in $\mathbb{P}_1$, since we can collect the conditions for failure together. For example, the following program:

$$\texttt{if } A_1 \texttt{ then } \perp \texttt{ else } P_1\texttt{; if } A_2 \texttt{ then } P_2 \texttt{ else if } A_2 \texttt{ then } \perp \texttt{ else } P_3$$

is equivalent to

$$\texttt{if } A_1 \vee (\neg A_2 \wedge A_2) \texttt{ then } \perp \texttt{ else } P_1\texttt{; if } A_2 \texttt{ then } P_2 \texttt{ else } P_3$$

The new program has size polynomial in the size of the old program, since the number of $\perp$ in the input program is bounded from above by the size of the input. Without loss of generality, suppose then that $\mathbb{P}_1$ has the form

$$\texttt{if } b \texttt{ then } P_1 \texttt{ else } \perp\texttt{; } \bar{\mathbb{P}}_1$$

where $\perp$ occurs only once in the program and $\bar{\mathbb{P}}_1$ has no failure statements. If the condition b is a disjunction of literals[2] then we can find a single polynomial B whose vanishing represents b. For example, if b is $(P_2 = 0) \vee \neg(P_3 = 0) \vee \neg(P_4 = 0)$, then we build the polynomial $B = P_2(t_3P_3 - 1)(t_4P_4 - 1)$. The new programs then become

[2] A *disjunction* is simply a boolean "OR" applied to several propositions. A *literal* is simply a variable, or the negation thereof.

$$\mathbb{P}_1' = P_1; \bar{\mathbb{P}}_1$$
$$\mathbb{P}_2' = \mathbb{P}_2; B; t_3(t_3P_3 - 1); P_3(t_3P_3 - 1); t_4(t_4P_4 - 1); P_4(t_4P_4 - 1)$$
$$\mathbb{Q}_1' = \mathbb{Q}_1$$
$$\mathbb{Q}_2' = \mathbb{Q}_2; B; t_3(t_3P_3 - 1); P_3(t_3P_3 - 1); t_4(t_4P_4 - 1); P_4(t_4P_4 - 1)$$

Here t_3 and t_4 are new random variables but they are uniquely determined by P_3 and P_4 under the constraints. Namely if $P_3 = 0$, then $t_3 = 0$, otherwise $t_3 = 1/P_3$. For a more general proposition formula b, we first convert it to a CNF formula,[3] which may result in a conjunction of exponentially many disjunctions, hence exponentially many polynomials $B_1, B_2, \cdots, B_m$, in addition to polynomials like $t_i(t_iP_i - 1)$ and $P_i(t_iP_i - 1)$ etc. The new equivalence is then

$$P_1; \bar{\mathbb{P}}_1 | \mathbb{P}_2, B_1, B_2, \ldots \approx \mathbb{Q}_1 | \mathbb{Q}_2, B_1, B_2, \ldots$$

Nevertheless, we have only introduced polynomially many new variables, since we need at most one new variable for each polynomial in the original program. Also, while the program $\mathbb{P}_2'$ may be exponentially long, the program $\mathbb{P}_1'$ is actually shorter than the original $\mathbb{P}_1$.

Observe that we may also assume that all the inputs to conditional statements are literals. For example we can replace

$$\texttt{if } A_1 \vee A_2 \texttt{ then } P_1 \texttt{ else } P_2$$

by

$$\texttt{if } A_1 \texttt{ then } P_1 \texttt{ else if } A_2 \texttt{ then } P_1 \texttt{ else } P_2.$$

Then, to remove "`if`" in a conditional statement such as

$$\cdots; \texttt{ if } \neg(B = 0) \texttt{ then } P_1 \texttt{ else } P_2; \cdots | \mathbb{P}_2$$

we can use classical tricks such as replacing disequalities by equalities with an extra variable to obtain

$$\cdots; P_2 + (tB)(P_1 - P_2); \cdots | \mathbb{P}_2; B(Bt - 1); t(Bt - 1)$$

Note that this step may increase the size exponentially, but the number of variables grows only polynomially.

The reduction we have just outlined is similar to the reduction [3] used to derive their doubly exponential algorithm to solve the general universal equivalence. Our version is slightly better since we do not introduce as many new variables.

Let us at last detail the key trick behind our improved zeta function algorithm.

3. Effective Kronecker theorem over finite fields

A classical theorem of Kronecker [9] says that any affine algebraic set defined by a system of m ($> n$) polynomials in n variables over an algebraically closed field K can be set-theoretically defined by a system of $n + 1$ polynomials in n variables over the same field K. Kronecker stated his theorem without a detailed proof; see [12] for a self-contained proof. The theorem, as stated, is actually true for any infinite field K, not necessarily algebraically closed. But it fails for certain finite fields (depending on the underlying polynomial system), which is our main concern here. Here we follow the ideas in [12] to show that Kronecker's theorem remains true for a finite field $\mathbb{F}_q$, if q is suitably large, and tailor our version to our algorithmic applications.

Recall that if I is an ideal in the commutative ring $\mathbb{F}_q[x_1, \ldots, x_n]$, then its *radical ideal* is defined as $\sqrt{I} = \left\{ f \in \mathbb{F}_q[x_1, \ldots, x_n] \,\middle|\, f^i \in I \text{ for some } i \geq 1 \right\}$. It is then clear that the two ideals I and $\sqrt{I}$ have the same set of $\mathbb{F}_{q^k}$-rational points for every k. In particular, they have the same zeta function.

[3] *Conjunctive Normal Form*, meaning "an AND of ORs"....

Theorem 3.1 *(Affine version). Let $f_i \in \mathbb{F}_q[x_1, \ldots, x_n]$ with $\deg(f_i) \le d$ for all $i \in \{1, \ldots, m\}$. Assume that $q > (n+1)d^n$. Then there is a deterministic algorithm with running time $m(nd^n)^{O(n)} \log^2 q$ or $O(\max^3\{m, n\} \log^2 q)$, according as $d \ge 2$ or $d = 1$, which finds $n+1$ polynomials $g_j \in \mathbb{F}_q[x_1, \ldots, x_n]$ with $\deg(g_j) \le d$ for all $j \in \{1, \ldots, n+1\}$ such that their radical ideals are the same: $\sqrt{(f_1, \ldots, f_m)} = \sqrt{(g_1, \ldots, g_{n+1})}$.*

We prove Theorem 3.1 after first proving the following homogeneous version:

Theorem 3.2 *(Homogeneous version). Let $f_i \in \mathbb{F}_q[x_1, \ldots, x_n]$ be homogeneous polynomials of degree d for all $i \in \{1, \ldots, m\}$ and assume $q > nd^{n-1}$. Then there is a deterministic algorithm, running in time $m(nd^n)^{O(n)} \log^2 q$ or $O(\max^3\{m, n\} \log^2 q)$, according as $d \ge 2$ or $d = 1$, which finds n homogeneous polynomials $g_j \in \mathbb{F}_q[x_1, \ldots, x_n]$ of degree d for all $j \in \{1, \ldots, n\}$ such that their radical ideals are the same: $\sqrt{(f_1, \ldots, f_m)} = \sqrt{(g_1, \ldots, g_n)}$.*

Proof of Theorem 3.2. If $m \le n$ then the theorem is trivial as we can just take $g_j = f_j$ for $j \le m$ and $g_j = f_1$ for $j > m$. So we assume $m > n$ henceforth. By induction, it is enough to prove the case $m = n+1$. The case $d=1$ is immediate from Gaussian Elimination, so let us also assume $d \ge 2$.

Now, the $n+1$ polynomials $\{f_1, \ldots, f_{n+1}\}$ in n variables are algebraically dependent over $\mathbb{F}_q$. That is, there is a non-zero homogeneous polynomial $A_M(y_1, \ldots, y_{n+1})$ of some positive degree M in $\mathbb{F}_q[y_1, \ldots, y_{n+1}]$ such that

$$A_M(f_1, \ldots, f_{n+1}) = \sum_{k_1 + \cdots + k_{n+1} = M} A_{k_1, \ldots, k_{n+1}} f_1^{k_1} \cdots f_{n+1}^{k_{n+1}} = 0.$$

This polynomial relation gives a homogeneous linear system over $\mathbb{F}_q$ with $\binom{M+n}{n}$ variables $A_{k_1, \ldots, k_{n+1}}$ and $\binom{Md+n-1}{n-1}$ equations (one equation for each monomial of degree dM in the variables $x_1, \ldots, x_n$). If $\binom{M+n}{n} > \binom{Md+n-1}{n-1}$ then the homogeneous linear system must have a non-trivial solution. So let us choose $M = nd^{n-1}$: Clearly $Md + i \le d(M+i)$ for all $i \ge 0$ and

$$\frac{\binom{M+n}{n}}{\binom{Md+n-1}{n-1}} = \frac{M+n}{n} \prod_{i=1}^{n-1} \frac{M+i}{Md+i} \ge \frac{M+n}{n} \left(\frac{1}{d}\right)^{n-1} > 1.$$

Solving our linear system takes time at most

$$O\left(\binom{M+n}{n}^{\omega} \log^2 q\right) = M^{O(n)} \log^2 q = (nd^n)^{O(n)} \log^2 q,$$

(with $\omega < 2.373$ the matrix multiplication complexity exponent [13]). So we can then clearly find a non-trivial solution involving $A_{k_1, \ldots, k_{n+1}} \in \mathbb{F}_q$ with $k_1 + \cdots + k_{n+1} = M$.

Next, we would like to make an invertible $\mathbb{F}_q$-linear transformation

$$y_u = \sum_{v=1}^{n+1} b_{u,v} z_v, \ b_{u,v} \in \mathbb{F}_q, \ u \in \{1, \ldots, n+1\}$$

such that when $A_M(y_1, \ldots, y_{n+1})$ is expanded as a polynomial in $z_1, \ldots, z_{n+1}$ under the above linear transformation, the coefficient of z_{n+1}^M is non-zero. Such an invertible linear transformation may not exist if q is small. We shall prove that it does exist if $q > M = nd^{n-1}$: Expand and write

$$A_M(y_1, \ldots, y_{n+1}) = \sum_{k_1 + \cdots + k_{n+1} = M} B_{k_1, \ldots, k_{n+1}} z_1^{k_1} \cdots z_{n+1}^{k_{n+1}}.$$

One easily checks that the coefficient of z_{n+1}^M is

$$\mathcal{B}(b_{1,n+1},\ldots,b_{n+1,n+1}) := B_{0,\ldots,0,M} = \sum_{k_1+\cdots+k_{n+1}=M} A_{k_1,\ldots,k_{n+1}} b_{1,n+1}^{k_1}\cdots b_{n+1,n+1}^{k_{n+1}},$$

and is an $(n+1)$-variate homogeneous polynomial in $\mathbb{F}_q[b_{1,n+1},\ldots,b_{n+1,n+1}]\backslash\{0\}$ of degree M. Since we assume $M<q$, the polynomial $\mathcal{B}$ is not the zero function on $\mathbb{F}_q^{n+1}$.

Now, since we have solved for the coefficients $A_{k_1,\ldots,k_{n+1}}$ of $A_M(y_1,\ldots,y_{n+1})$ already, we know the monomial term expansion for the homogeneous polynomial $\mathcal{B}_n \in \mathbb{F}_q[b_{1,n+1},\ldots,b_{n,n+1}]$ that satisfies $\mathcal{B} = \mathcal{B}_n(b_{1,n+1},\ldots,b_{n,n+1}) b_{n+1,n+1}^{d_{n+1}} + o(b_{n+1,n+1}^{d_{n+1}})$ where d_{n+1} is the degree of $\mathcal{B}$ in $b_{n+1,n+1}$, $\mathcal{B}_n$ is not the zero polynomial, and the second term means a sum of terms of degree strictly lower than d_{n+1} with respect to $b_{n+1,n+1}$. In particular, if $n=1$, we can simply pick $b_{1,2}$ to be any nonzero element of $\mathbb{F}_q$, and try $\leq d_2+1 \leq M+1$ elements of $\mathbb{F}_q$ until we find a $b_{2,2}$ making $\mathcal{B}(b_{1,2},b_{2,2})$ nonzero as desired.

Otherwise, if $n \geq 2$, we similarly determine the monomial term expansion for the homogeneous polynomial $\mathcal{B}_{n-1} \in \mathbb{F}_q[b_{1,n+1},\ldots,b_{n-1,n+1}]$ that satisfies

$$\mathcal{B}_n = \mathcal{B}_{n-1}(b_{1,n+1},\ldots,b_{n-1,n+1}) b_{n,n+1}^{d_n} + o(b_{n,n+1}^{d_n}),$$

and so on, to define successive leading coefficient polynomials $\mathcal{B}_{n-2},\ldots,\mathcal{B}_1$ in fewer and fewer variables. We then see that upon picking any nonzero element of $\mathbb{F}_q$ for $b_{1,n+1}$, and then checking $\leq M+1$ possible values for $b_{2,n+1}$, and $\leq M+1$ possible values for $b_{3,n+1}$, etc., we can make $\mathcal{B}(b_{1,n+1},\ldots,b_{n+1,n+1})$ nonzero after $(M+1)n$ polynomial evaluations.

Recall that powers like a^k in $\mathbb{F}_q$ can be evaluated via the binary method using $O(\log k)$ multiplications in $\mathbb{F}_q$. Recall also that multiplications in $\mathbb{F}_q$ can be done within time $O(\log^2 q)$, and even faster if Fast Fourier Transforms are used. (See, e.g., [2].) Observe then that each evaluation of $A_M(z_1,\ldots,z_{n+1})$ at a choice of $(b_{1,n+1},\ldots,b_{n+1,n+1})$ thus takes time $O(\binom{M+n}{n} n\log(M)\log^2 q) = (nd^{n-1})^{O(n)} n\log(nd^{n-1})\log^2 q = n^{O(n)} d^{O(n^2)} \log^2 q$. So then, multiplying our last bound by $(M+1)n = (nd^{n-1}+1)n$, we see that finding our desired vector $(b_{1,n+1},\ldots,b_{n+1,n+1})$ takes overall time $n^{O(n)} d^{O(n^2)} \log^2 q$.

The non-zero vector $(b_{1,n+1},\ldots,b_{n+1,n+1})$ can be easily extended to an invertible square matrix $(b_{u,v}) \in \mathrm{GL}_{n+1}(\mathbb{F}_q)$. For instance, if $b_{n+1,n+1} \neq 0$, then we can simply take $b_{u,v}=0$ for $u \neq v$ and $v \in \{1,\ldots,n\}$, and $b_{u,v}=1$ if $u=v\leq n$. More generally, we can simply do the same construction (up to a permutation of indices) if $b_{n+1,n+1}=0$ and we find some other $b_{i,n+1}$ that is nonzero. In this way, we obtain the desired invertible transformation.

To conclude, we substitute $f_i := \sum_{j=1}^{n+1} b_{i,j} g_j$ (thus actually defining the g_j as linear combinations of the f_i, since the matrix $[b_{i,j}]$ is invertible by construction) and rewrite our established polynomial relation in the form

$$A_M(f_1,\ldots,f_{n+1}) = c g_{n+1}^M + G_1(g_1,\ldots,g_n) g_{n+1}^{M-1} + \cdots + G_M(g_1,\ldots,g_n) = 0,$$

where c was our constructed nonzero value for $\mathcal{B}(b_{1,n+1},\ldots,b_{n+1,n+1})$ and $G_i(g_1,\ldots,g_n)$ is a homogeneous polynomial in $(g_1,\ldots,g_n)$ of degree i for $i \in \{1,\ldots,M\}$. Since $c \neq 0$ we deduce that $g_{n+1}^M \in (g_1,\ldots,g_n)$. It follows that

$$\sqrt{(f_1,\ldots,f_{n+1})} = \sqrt{(g_1,\ldots,g_{n+1})} = \sqrt{(g_1,\ldots,g_n)}$$

so we have proved the case $m=n+1$. Our general complexity bound can be attained simply by repeating the preceding argument on the new system $(g_1,\ldots,g_n,f_{n+2})$ and proceeding inductively. In other words, our final overall complexity bound is no more than m times $(nd^n)^{O(n)}\log^2 q$. ■

Proof of Theorem 3.1. We first homogenize the polynomials $f_1,\ldots,f_m$ to obtain a homogeneous polynomial $F_i(x_0,x_1,\ldots,x_n) := x_0^d f_i(x_1/x_0,\ldots,x_n/x_0)$ of degree exactly d for each $i \in \{1,\ldots,m\}$. Note that $F_i(1,x_1,\ldots,x_n) = f_i(x_1,\ldots,x_n)$ for all i.

Now applying Theorem 3.2 to $\{F_1,\ldots,F_m\}$ we obtain a homogeneous polynomial $G_j(x_0,x_1,\ldots,x_n)$ of degree d, for each $j \in \{1,\ldots,n+1\}$, such that $\sqrt{(F_1,\ldots,F_m)} = \sqrt{(G_1,\ldots,G_{n+1})}$. Setting $x_0=1$

in the last equality one obtains $\sqrt{(f_1,\ldots,f_m)} = \sqrt{(g_1,\ldots,g_{n+1})}$ where we define $g_i(x_1,\ldots,x_n) := G_i(1,x_1,\ldots,x_n)$. ■

4. The computation of zeta functions: proving Theorem 1.1

Let F be the following polynomial system with m equations and n variables over $\mathbb{F}_q$:

$$F(x_1,\ldots,x_n) = (f_1(x_1,\ldots,x_n),\ldots,f_m(x_1,\ldots,x_n)),$$

where each $f_i \in \mathbb{F}_q[x_1,\ldots,x_n]$ has total degree at most d. To compute the zeta function $Z(F,T)$, we need the following explicit degree bound of Bombieri:

Lemma 4.1. *[5] The total degree (the sum of the degrees of the numerator and denominator) of the zeta function $Z(F,T)$ is no greater than $(4d+5)^{2n+1}$.* ■

Note that this total degree bound is independent of m. This already suggests the possibility of improving the dependence on m in earlier algorithms for computing zeta functions. By applying our effective Kronecker theorem (Theorem 3.1), we are now ready to prove our main result.

Proof of Theorem 1.1. If $q > (n+1)d^n$ then we can apply the affine effective Kronecker theorem in the previous section to replace the large polynomial system F by a smaller polynomial system $G = (g_1(x_1,\ldots,x_n),\ldots,g_{n+1}(x_1,\ldots,x_n))$, where each $g_j \in \mathbb{F}_q[x_1,\ldots,x_n]$ has total degree at most d. The smaller system G can be constructed in time

$$m(nd^n)^{O(n)}\log^2 q,$$

thanks to Theorem 3.1. The two systems F and G have the same number of solutions over every extension field $\mathbb{F}_{q^k}$. In particular, their zeta functions are the same, namely, $Z(F,T) = Z(G,T)$. Now, by the algorithms in [8], the zeta function $Z(G,T)$ can be computed in time

$$2^{n+1}p((n+1)^n d^n \log q)^{O(n)} = p(n^n d^n \log q)^{O(n)}.$$

Thus, the zeta function $Z(F,T)$ can be computed in time

$$m(nd^n)^{O(n)}\log^2(q) + p(n^n d^n \log q)^{O(n)} = d^{O(n^2)}(mn^{O(n)}\log^2(q) + p(n^n \log q)^{O(n)}).$$

If $q \le (n+1)d^n$ then we cannot apply our effective Kronecker theorem directly. So we use a somewhat different argument instead: Let $B = (4d+5)^{2n+1}$ be the upper bound in Bombieri's lemma. By Corollary 2.7 and its proof in [14], it is enough to compute the following B numbers

$$N_{q^k}(F);\ k \in \{1,\ldots,B\}.$$

If $q^k \le (n+1)d^n$, namely, $k \le \log((n+1)d^n)/\log q$, then we simply use exhaustive search to compute $N_{q^k}(F)$. For each such k this takes time

$$q^{kn}m(d^n\log q)^{O(1)} \le ((n+1)d^n)^n m(d^n\log q)^{O(1)} = m(n+1)^n d^{O(n^2)}(\log q)^{O(1)}.$$

If $q^k > (n+1)d^n$, namely, $\log((n+1)d^n)/\log q < k \le B$, then we can apply Theorem 3.1 to the system over the extension field $\mathbb{F}_{q^k}$ to produce a new system

$$G_k = (g_{k,1}(x_1,\ldots,x_n),\ldots,g_{k,n+1}(x_1,\ldots,x_n)),$$

where each $g_{k,j} \in \mathbb{F}_{q^k}[x_1,\ldots,x_n]$ has total degree at most d. Note that this takes time $m(nd^n)^{O(n)}\log^2(q^k) = mk^2(nd^n)^{O(n)}\log^2 q \le mB^2(nd^n)^{O(n)}\log^2 q = md^{O(n)}(nd^n)^{O(n)}\log^2 q = m(nd^n)^{O(n)}\log^2 q$. Now,

$$N_{q^k}(F) = N_{q^k}(G_k).$$

The system has only $n+1$ equations and thus the number $N_{q^k}(G_k)$ (in fact the full zeta function of G_k over $\mathbb{F}_{q^k}$) can be computed by [8] in time

$$2^{n+1}p(k(n+1)^n d^n \log q)^{O(n)} = p(Bn^n d^n \log q)^{O(n)} = p(n^n d^n \log q)^{O(n)}.$$

Thus, the total time to compute $Z(F,T)$ is bounded from above by

$$m(nd^n)^{O(n)} \log^{O(1)}(q) + Bmp(n^n d^n \log q)^{O(n)} = d^{O(n^2)}(mn^{O(n)} \log^{O(1)}(q) + p(n^n \log q)^{O(n)}). \quad \blacksquare$$

Acknowledgements

We thank the referees for their sharp-eyed comments that greatly helped improve our writing. We also gratefully acknowledge the US National Science Foundation for their support through grants CCF-1900820 (Cheng), CCF-1900881 (Rojas), and CCF-1900929 (Wan).

References

[1] Martín Avendaño, Ashraf Ibrahim, J. Maurice Rojas, Korben Rusek, Faster p-adic feasibility for certain multivariate sparse polynomials, in: Special Issue in Honor of 60th Birthday of Joachim von zur Gathen, J. Symb. Comput. 47 (4) (April 2012) 454–479.
[2] Eric Bach, Jeff Shallit, Algorithmic Number Theory, Efficient Algorithms, vol. I, MIT Press, Cambridge, MA, 1996.
[3] Gilles Barthe, Charlie Jacomme, Steve Kremer, Universal equivalence and majority of probabilistic programs over finite fields, in: LICS '20: Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, July 2020, pp. 155–166.
[4] Jingguo Bi, Qi Cheng, J. Maurice Rojas, Sub-linear root detection, and new hardness results, for sparse polynomials over finite fields, SIAM J. Comput. 45 (4) (2016) 1433–1447.
[5] Enrico Bombieri, On exponential sums in finite fields, II, Invent. Math. 47 (1988) 29–39.
[6] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, Frederik Vercauteren (Eds.), Handbook of Elliptic and Hyperelliptic Curve Cryptography, Discrete Mathematics and Its Applications (Boca Raton), Chapman & Hall/CRC, Boca Raton, FL, 2006.
[7] Joachim von zur Gathen, Marek Karpinski, Igor Shparlinski, Counting curves and their projections, Comput. Complex. 6 (1) (1996–1997) 64–99.
[8] David Harvey, Computing zeta functions of arithmetic schemes, Proc. Lond. Math. Soc. 111 (6) (2015) 1379–1401.
[9] Leopold Kronecker, Grundzüge einer arithmetischen Theorie der algebraischen Grossen, J. Reine Angew. Math. 92 (1882) 1–123.
[10] Shuvendu K. Lahiri, Andrzej Murawski, in: Ofer Strichman, Mattias Ulbrich (Eds.), Program Equivalence, Report from Dagstuhl Seminar, 1815, pp. 8–13, http://www.dagstuhl.de/18151, 2018.
[11] Alan Lauder, Daqing Wan, Counting rational points on varieties over finite fields of small characteristic, in: Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography, in: Math. Sci. Res. Inst. Publ., vol. 44, Cambridge Univ. Press, Cambridge, 2008, pp. 579–612.
[12] Oscar Perron, Beweis und Verschärfung eines Satzes von Kronecker, Math. Ann. 118 (1941–1943) 441–448.
[13] Virginia Vassilevska-Williams, Limits on all known (and some unknown) approaches to matrix multiplication, in: Proceeding of International Symposium on Symbolic and Algebraic Computation, ISSAC 2019, Beijing, China, 2019.
[14] Daqing Wan, Algorithmic theory of zeta functions over finite fields, in: Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography, in: Math. Sci. Res. Inst. Publ., vol. 44, Cambridge Univ. Press, Cambridge, 2008, pp. 551–578.