AN EFFECTIVE DESCRIPTION OF THE ROOTS OF MULTIVARIATES MOD p^k AND THE RELATED IGUSA'S LOCAL ZETA FUNCTION

SAYAK CHAKRABARTI AND NITIN SAXENA

ABSTRACT. Finding roots of a polynomial $f(x_1, \ldots, x_n)$, in a prime field $\mathbb{F}_p$, is a basic question; with a long history and several practical algorithms known. This question is open when we ask it for roots mod p^k, $k \geq 2$. This is due to the difficulty of lifting *singular* $\mathbb{F}_p$-roots to $\mathbb{Z}/\langle p^k \rangle$. Egs. $f = x_1^3 - px_2^2$, or $f = x_1^3 - p$, modulo p^2. Now, it is unclear how to even test the *existence* of roots; as the only $\mathbb{F}_p$-root here, $x_1 \equiv 0 \bmod p$, starts behaving unpredictably when 'lifted' modp^2.

In this work, we give the first algorithm to describe these roots in a practical way. Notably, when n & degree of f are constant, our deterministic algorithm is polynomial-time (in k, p). Our method gives the first efficient algorithms for the following problems —which were erstwhile open even for $n = 2$— (1) to represent all (∞-many) p-adic roots, and (2) to compute Igusa's local zeta function. In general, ours is a new effective method to prove the rationality of this zeta function.

Keywords. polynomial, prime-power, multivariate, p-adic, root-finding, root-counting, deterministic, Igusa, zeta function, tree

CONTENTS

1. INTRODUCTION

Roots of polynomials in fields and rings have played an important role in mathematics and computer science for decades, with applications in a variety of topics. The roots of univariate polynomials modulo prime powers are relatively easy to find as we can find the roots in finite fields using factorization [CZ81, Ber67, Ber70, KU11, Rón87], after which we can efficiently lift the roots to modulo higher powers of p using the recent developments from [Pan95, BLQ13, DMS21]. However, even though we can factorize *multi*variate polynomials [Kal82, Kal85], their roots usually do not correspond to a factor. Eg. even the irreducible polynomial $y-x$ has numerous roots! Such polynomials pose problems in root-finding as they have exponentially many roots in the base/prime field itself, and we can not quickly guess which root will lift to $\bmod p^2$. In our paper, we resolve this issue by giving an efficient algorithm to find roots modulo p^k for any p and k (given in unary); assuming that the degree d of the polynomial and the number n of the variables, are both small.

Prime field $\mathbb{F}_p$ and *p-adic integers* $\mathbb{Z}_p$ are unique factorization domains, and polynomials (above them) behave in an expected way. There are some algorithms to factorize polynomials in $\mathbb{Z}_p$ [Chi87, CG00, GNP12]. However, the properties in rings of characteristic as prime powers, which can be seen as a world between $\mathbb{F}_p$ and $\mathbb{Z}_p$, are still a mystery to us. There has been extensive work in this since the famous Hensel's lifting [Hen18, Zas69, Zas78], where factors of polynomials are lifted from $\mathbb{F}_p$ to $\mathbb{Z}/p^k\mathbb{Z}$. Several variants of Hensel's lifting are available in various topics in algebra & number theory; but they fail when the polynomial does not factorize into coprime factors. This, when interpreted in terms of roots, means that it is difficult to lift *singular* roots (i.e. of 'multiplicity' ≥ 2, as we will see later). There have been partially successful attempts to tackle this, and achieve factorization of univariate polynomials $\bmod p^k$ [DMS21, Sir17, Săl05, CL01]. Factorization $\bmod p^k$ has only been solved until $k \leq 4$. Due to this, we can not use factorization, in any way, when finding roots $\bmod p^k$. Furthermore, due to the availability of 'exponentially' many factors, as well as roots, in these rings, its (data) structure has been of interest in mathematics and computer science. Eg. [DM97, Mau01, GCM21] analyze root-sets $\bmod p^k$, while [DMS19, CGRW19, KRRZ20, DS20, RRZ21] count the number of roots for a given polynomial. However, most of the works are restricted to univariate polynomials, as we did not have practical algorithms to find even *one* root of *bi*variate polynomials $\bmod p^k$. *This paper gives the first algorithm to find all the roots of an n-variate degree-d polynomial, over $\mathbb{Z}/p^k\mathbb{Z}$ resp. $\mathbb{Z}_p$, efficiently (for fixed d, n and varying p, k)*, thus paving the way for many other problems.

Let us take a famous example as our algorithm's special-case— *Elliptic curves* —they have been of great interest to mathematicians and computer scientists over the past century (egs. integer factoring [LJ87, Bre86], cryptography [Mil85], discrete logarithms [Gau00]). An elliptic curve can be seen as the bivariate polynomial $f(x,y) = y^2 - x^3 - ax - b$. Similarly, its generalization, a *hyperelliptic curve*, is given by the equation $y^2 + u(x)y + v(x) = 0$. Rational roots of (hyper)elliptic curves have been widely studied with several papers in this area [Sch95, MVZ93, LL03, LMMS94, GH00, Ked01, Sat02, MCT02]. Our algorithm can find all ($\mathbb{Z}/p^k\mathbb{Z}$)-roots (resp. p-adic) of not only these, but general curves (that may have singularities).

Similarly, our algorithm finds certain roots of a fixed *Diophantine equation*, a question that has been of interest to mathematicians for ages. A Diophantine equation is a polynomial equation, whose solutions required are usually in $\mathbb{Z}$ or $\mathbb{Z}/N\mathbb{Z}$. If N is composite, we can find roots modulo prime powers using our algorithm, and then use Chinese Remainder Theorem to find the solution modulo N. Decidability of Diophantine equations is Hilbert's 10th problem; famously, [DPR61, Mat70] showed that computably enumerable sets can be written as a Diophantine equation.

Root finding and counting have interesting applications in complexity theory too. We know that finding a solution to a system of constant-degree polynomials in any ring is NP-complete. Our algorithm solves a special case: where we want a common solution of *low*-variate, *low*-degree polynomials. Furthermore, root counting is a very hard problem. [EK90] showed that counting the number of roots of a multivariate polynomial of degrees as small as 3 is $\#\mathsf{P}$-complete, while [GGL08] showed that modular root counting, over $\mathbb{F}_q$, is NP-hard for prime modulus other than the characteristic of the field. [ZG03] also showed that the problem of computing Igusa's local zeta function is NP-complete even for $p = d = 2$ which can be shown from the arithmetization of SAT.

There are more applications of roots of multivariate polynomials in different aspects of computer science. [DMS21] reduced the problem of univariate-factoring in these rings, to finding roots of a *bi*variate polynomial. In error-correcting codes, list-decoding of Reed-Solomon codes requires finding roots of a bivariate polynomial in $\mathbb{F}_q[x_1, x_2]$ [GS98, Sud96]. Also, root finding of certain polynomials in $\mathbb{F}_q$ is equivalent to maximum-likelihood decoding of Reed-Solomon codes [GGL08]. There have been some generalizations of list-decoding of Reed-Solomon codes to Galois rings, $\mathbb{G} = \mathbb{Z}[x]/\langle p^k, \phi(x)\rangle$ [DMS19, McD74, LN94]. More literature on error-correcting codes in Galois rings can be found in [Sud97, CHJK$^+$94, BLQ13, BW10]. Roots of multivariates also have several applications in cryptography, which can be found in [BGGI09, Cop96b, Cop96a, Cop97, Cor04].

Root finding, and counting, of polynomials $\bmod p^k$ appears in analytic number theory and arithmetic geometry [NZM91, Apo98]. An important topic that appears in mathematics is the theory of zeta functions. Zeta functions have played a crucial role in mathematics and have applications in diverse fields. The most important zeta function is the Reimann zeta function [Rie59], which encodes the distribution of primes [Con03, THB86]. [Rie59] used it to obtain a better approximation bound on the prime number theorem, while several other applications of Riemann zeta function have been developed in mathematics and other sciences. More work on zeta functions can be found in [Apo10].

There has been extensive work on computing related zeta functions. [Ked04, Lau06] computes the zeta function which encodes the size of a variety in finite fields and extensions, in time polynomial in the characteristic p. Improving this, say to $\mathsf{poly}(\log p)$, is a central open question. Our paper too focuses on the regime of $\mathsf{poly}(p)$-time. Recently, [Har15, CRW20] new algorithms on computing this zeta function on special cases, however still in time $\mathsf{poly}(p)$.

In this paper, we are interested in another zeta function called the *Igusa's local zeta function* (Igusa's LZF), used to encode the number of roots of a polynomial modulo prime powers. *Local* implies that it is associated with p-adic analysis.

Formally, for a polynomial $f(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$, the Igusa's local zeta function is defined as

$$Z_{f,p}(s) := \int_{\mathbb{Z}_p^n} |f(\mathbf{x})|_p^s \cdot |d\mathbf{x}|, \tag{1}$$

for $s \in \mathbf{C}$, where the real part of s is positive.

Igusa first proved that this function $Z_{f,p}(s)$ converges to a rational function. Followed by this, [Den84] gave another proof for the rationality of this function. However, both the proofs of Igusa and Denef were quite complicated and we give a much simpler proof by computing the Poincaré series, which can be seen as a generating function for $Z_{f,p}(s)$. The Poincaré series for a polynomial f and a prime p is defined as

$$P_{f,p}(t) := \sum_{i=0}^{\infty} N_i(f) \cdot (p^{-n}t)^i, \tag{2}$$

where $t \in \mathbb{C}$, $|t| < 1$. Choosing $t = p^{-s}$, we denote $P_{f,p}(p^{-s}) = \sum_{i=0}^{\infty} N_i(f) \cdot (p^{-n}p^{-s})^i$. [Igu07] showed a connection between Poincaré series and Igusa's LZF given as:

$$P_{f,p}(p^{-s}) = \frac{1 - p^{-s} \cdot Z_{f,p}(s)}{1 - p^{-s}}.$$

Notice that proving rationality of Poincaré series $P(p^{-s})$ in p^s implies the rationality of Igusa's LZF in p^s and vice versa. We use this relation to show that Poincaré series is indeed a rational function by counting roots modulo p^k.

Despite the rationality being proved, explicit computation of this zeta function has remained a challenge. Root counting helps in computing this using the Poincare series [DS20, ZG03, DH01], but has been restricted to univariates. Giving the first algorithm, *we compute Igusa's local-zeta function for bivariates; thus, giving a new proof of its rationality as well.*

1.1. **Previous work.**

There have been several works on finding roots of polynomials in rings. [Pan95] gave an approach for finding roots of univariate polynomials modulo prime-powers in randomized polynomial time, which was greatly improved by [BLQ13]. [NRS17] further improved the time complexity of [BLQ13] to find roots in these rings. We are aware of only one work studying roots of bivariate polynomials mod p^k [RRZ21], where they count the total number of roots if the given polynomial $f(x, y)$ can be written as $f_1(x) + f_2(y)$, i.e. variable 'separated'. On the other hand, our algorithm does not require such assumptions.

Roots modulo p^k can be seen as an intermediate-world between roots in $\mathbb{F}_p$ and roots in $\mathbb{Z}_p$. There have been papers to find roots of system of polynomials in certain finite fields [HW99, LPT$^+$17, BKW19, Kay05]. However, for finding $\mathbb{Z}_p$ roots, certain upper bounds given by N have been shown in [BM67, Chi21] which state that the existence of a solution mod p^N implies a $\mathbb{Z}_p$-root. Among the improved results, [Chi21] showed $N \leq (nd)^{O(n^3 2^n)}$. [DS20] showed $N \leq d(\Delta + 1)$, where Δ is the discriminant-valuation; but it is only for univariate polynomials. We prove stronger structural results for the p-adic-roots of multivariate polynomials, as discussed in Section 5.1.

Clearly, the literature suggests that roots behave "nicely" in $\mathbb{F}_p$ and $\mathbb{Z}_p$; but the properties in $\mathbb{Z}/p^k\mathbb{Z}$ are quite different for 'small' $k \geq 2$. In a way, our paper generalizes the approach of [BLQ13] to nontrivially *reduce* root finding modulo p^k to the problem of solving a multivariate multi-polynomial system in lesser number of variables.

Computation of Igusa's local zeta function, and its rationality via the Poincaré series, have also been an important question in computational number theory. [Igu74, Igu77] proved the rationality of the Poincaré series, while [Den84] proved the same for a system of polynomials. [DS20] gave the first algorithm to compute Igusa's local zeta function of univariates in (deterministic) poly-time. In this paper, by describing the p-adic roots of multivariates, we give the first algorithm to compute Igusa's local zeta function practically, as well as re-prove the rationality of the Poincaré series.

1.2. **Our results: Find roots in $\mathbb{Z}/p^k\mathbb{Z}$, $\mathbb{Z}_p$, and compute the Poincaré series.**

Our results give a new constructive understanding of the roots modulo p-power of multivariate systems. We use a new data-structure in the form of a *tree*, to view these roots with increasing exponent of the modulus. This tree data-structure, essentially, performs *desingularization* of roots— segregating them until they are non-singular —and lift to the required exponent of the prime. Furthermore, we devise a new data-structure called *representative roots* to represent the exponentially (or p-adically infinite) many roots compactly.

Theorem 1.1 (Bivariates)**.** *Given a polynomial $f(x_1, x_2) \in \mathbb{Z}[x_1, x_2]$ of degree d, we can decide if a root of $f(x_1, x_2) \bmod p^k$ exists, in deterministic $\mathsf{poly}((k + d + p)^d)$ time. If roots do exist, we can find and count all the roots (i.e. output them in a compact data structure).*

Based on Theorem 1.1, we give the following two corollaries, both of which were open problems. In a way, we bridge the gap between rings of the form $\mathbb{Z}/p^k\mathbb{Z}$ and $\mathbb{Z}_p$ by giving better bounds than existing works. Using this, we efficiently compute Igusa's local zeta function.

Corollary 1.2 (p-adic)**.** *Given a polynomial $f(x_1, x_2) \in \mathbb{Z}[x_1, x_2]$ of degree d and the absolute value of its coefficients bounded above by $M > 0$, we can find all the p-adic-roots of f (in $\mathbb{Z}_p$) in deterministic $\mathsf{poly}((\log M + d + p)^d)$ time (i.e. output their k digits in a compact data structure).*

Corollary 1.3 (Local-zeta fn.)**.** *Given a polynomial $f(x_1, x_2) \in \mathbb{Z}[x_1, x_2]$ of degree d and the absolute value of its coefficients bounded above by $M > 0$, we can compute the Poincaré series $P(t) =: A(t)/B(t)$ associated with f and a prime p, in deterministic $\mathsf{poly}((\log M + d + p)^d)$ time.*

There are major dissimilarities between roots of univariate and bivariate polynomials. However, n-variate polynomials, for 'small' $n \geq 2$, behave in a roughly similar manner in our proof. Thus, after describing root-finding of bivariates (Theorem 1.1), we sketch its generalization to n-variates.

Theorem 1.4 (n-variates)**.** *Given a polynomial system* $\{f_i(x_1, \ldots, x_n) \in \mathbb{Z}[\mathbf{x}] \mid \forall i \in [m]\}$ *of degrees* $\leq d$ *and the absolute value of its coefficients bounded above by* $M > 0$*. We can find all its* common $(\mathbb{Z}/p^k\mathbb{Z})$*-roots, resp. its* p*-adic-roots, resp. its Poincaré series, in deterministic* $\mathsf{poly}((m \log M + d + p)^{(2d(n-1))^{n-1}})$ *time.*

1.3. **Difficulty of the problem.** The main focus of this paper is to find roots of a multivariate polynomial $\bmod\, p^k$. There do exist methods to find roots of multivariates in $\mathbb{F}_p$ resp. $\mathbb{Z}_p$. Many algorithms require time exponential in $\log p$ [LPT+17, Chi21, HW99]. As we mentioned, arithmetic in rings of the form $\mathbb{Z}/p^k\mathbb{Z}$ is more difficult than in $\mathbb{F}_p$ or $\mathbb{Z}_p$. Thus, the rings $\mathbb{Z}/p^k\mathbb{Z}$ have several open problems, root finding of a multivariate being one of them. This paper addresses this issue and introduces a datastructure that returns *all* the roots.

It is easy to lift a *non-singular* $\mathbb{F}_p$-root, i.e. root of $f \bmod p$ at which some first-order derivative of f is nonzero, to any $\mathbb{Z}/p^k\mathbb{Z}$ (see Theorem 2.1 and Corollary 2.4). In contrast, our algorithm is a significant advancement in the case when $\mathbb{F}_p$-roots are *singular.* It can be viewed as a reduction to non-singular roots, of a 'higher' dimension variety: which gets created while using the process of *lifting* (eg. extend a root (x_1, x_2) of $f \bmod p^j$ to $\bmod\, p^{j+1}$). In our method, the dependence on p cannot be improved, as bivariates can have $O(p)$ many roots at each step of lifting.

Practicalities. Though this algorithm is slow for large degree *bi*variates or large primes, it is the first idea that works, for *small* degree d and prime p, much better than the brute-force algorithm. Our general algorithm is doubly-exponential in n (= number of variables), but, unsurprisingly the complexity is expected to be 'bad' in n as the problem of counting $\mathbb{F}_2$-roots (say, for a system with $d = 2$) is already NP-hard and even $\#\mathsf{P}$-hard [EK90, ZG03, GGL08].

1.4. **Proof overview: Root-Find().** We prove Theorem 1.1 by giving an algorithm that returns all the possible roots modulo p^k, of a given degree-d polynomial $f(x_1, x_2) \in \mathbb{Z}[x_1, x_2]$. It designs a compact data structure for this, and outputs what we call *representatives.*

The root-finding is done iteratively: find each p-adic coordinate which is a root at each step, and perform lifting to higher coordinates; as is done in the case of univariates [BLQ13, DMS19, DMS21, NRS17]. If (a_1, a_2) is a root of $f(x_1, x_2) \bmod p$, then we transform the polynomial to another polynomial given by $f(a_1 + px_1, a_2 + px_2)$. In order to find $\mathbb{F}_p$-roots of this polynomial in the next step, we remove the 'extra' p-powers by dividing by p^v; where $v := v_p(f(a_1 + px_1, a_2 + px_2))$ is the 'val-multiplicity' and $v_p(\cdot)$ is the p-valuation (Definition A.2). We define this step, of transforming the coordinates and subsequent division by the p-power, as the *lifting step* or *lifting of roots.* The polynomial will be modified at each step such that its $\mathbb{F}_p$-root returns a coordinate of the final (p-adic resp. $\mathbb{Z}/p^k\mathbb{Z}$) root. Notice that if (a_1, a_2) is an $\mathbb{F}_p$-root of $f(x_1, x_2)$, and after lifting, the polynomial becomes $\tilde{f}(x_1, x_2) := p^{-v} f(a_1 + px_1, a_2 + px_2)$ which has an $\mathbb{F}_p$-root (b_1, b_2), then $(a_1 + pb_1, a_2 + pb_2)$ is a root of $f(x_1, x_2) \bmod p^{v+1}$. The *uni*variate case of this lifting technique, as developed in [BLQ13, DMS21], is explained in Section A.

However, it might so happen that root (a_1, a_2) in the lifting process does *not* lift to higher powers of p; but some other root does lift, as illustrated by the following example.

Example 1.5. *Consider* $f(x_1, x_2) := x_1^3 - x_2^3 + 3x_2 - 3x_1 + 5$ *and* $p := 5$*.* $(1, 1)$ *and* $(2, 2)$ *are its* $\mathbb{F}_p$*-roots. Starting with the root* $(1, 1)$*, the process of lifting given by the transformation* $(x_1, x_2) \mapsto (1 + 5x_1, 1 + 5x_2)$ *and division by* 5*, yields the polynomial* $25x_1^3 - 25x_2^3 + 15x_1^2 - 15x_2^2 + 1$ *which does not have* $\mathbb{F}_5$*-roots. Although, restarting with the root as* $(2, 2)$ *yields the polynomial* $25x_1^3 - 25x_2^3 + 30x_1^2 - 30x_2^2 + 9x_1 - 9x_2 + 1$ *after lifting.* $(1, 0)$ *is now its* $\mathbb{F}_5$*-root! This anomaly is explained by a curious fact:* $(1, 1)$ *is a singular root while* $(2, 2)$ *is non-singular.*

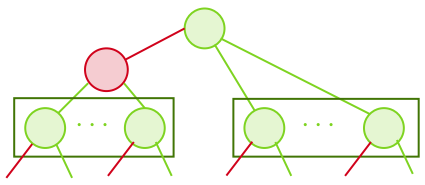

FIGURE 1. Branching along the tree

Thus, we iteratively loop over all the possible roots at each step, by fixing one variable, say x_1, with p-many possibilities, and finding the possible d-many (or p-many) values of x_2.

Val-multiplicity vs valuation. For a polynomial $f(\mathbf{x})$, we define the *effective polynomial* as $f(\mathbf{x}) \bmod p$, where the coefficients are in $\mathbb{F}_p$ (w.l.o.g. $f(\mathbf{x}) \bmod p$ is non-constant). Similarly, the *effective degree* of $f(\mathbf{x})$ is the degree of $f(\mathbf{x}) \bmod p$. Unless specified otherwise, we will denote $d_1 \geq 1$ as the effective degree of the polynomial at that step of lifting, while $d \geq d_1$ will be the total degree. ($d_1 = 0$ is trivial to handle.)

We define a *local root* of $f(\mathbf{x})$ as a root of the effective polynomial $f(\mathbf{x}) \bmod p$. For a local root $\mathbf{a} \in \mathbb{F}_p^2$, *local valuation* is defined as $v_p(f(\mathbf{a}))$. Similarly, *val-multiplicity of local root* is defined as $v_p(f(\mathbf{a} + p\mathbf{x}))$, i.e., the minimum valuation of the coefficients of the polynomial thus formed; we sometimes shorten it to val-mult($\mathbf{a}$). Obviously, val-multiplicity is at most the local valuation.

Branching w.r.t. val-multiplicities. As we have seen in Example 1.5, different $\mathbb{F}_p$ roots in steps of lifting can give rise to different val-multiplicities. Thus, we create a *tree* (Fig.1), having nodes as polynomials, say $f_j(\mathbf{x})$ in the j-th step of lifting, and branches arising from it corresponding to each local root $\mathbf{a} \in \mathbb{F}_p^2$. The children of this node will be the polynomials obtained from lifting by the root corresponding to the branch, given by $f_{j+1}(\mathbf{x}) := p^{-v} f_j(\mathbf{a} + p\mathbf{x})$; where $v :=$ val-mult($\mathbf{a}$).

Theorem 2.1-(1) shows that val-multiplicity is at most the effective degree (denoted d_1). Since each local root corresponds to some val-multiplicity in $[d_1]$, we associate these roots with their val-multiplicities. The branches of roots with the same val-multiplicity v, yield similar properties on the structure of the polynomial after lifting. So, we denote them as a single 'thick' val-multiplicity v branch (but they are in fact computed in several parallel val-multiplicity v branches and their corresponding polynomials as nodes). Note: There are at most p^2 local roots in $\mathbb{F}_p^2$.

The recursive steps of finding each precision coordinate can thus be seen as a tree, with each branch/child seen as a root $\mathbf{a} \in \mathbb{F}_p^2$ of val-multiplicity in $[d_1]$. If $f_j(x_1, x_2) \bmod p$ is a d_1-form at $\mathbf{a}$, then it is in the ideal $\langle x_1 - a_1, x_2 - a_2 \rangle^{d_1}_{\mathbb{F}_p[\mathbf{x}]}$ (Lemma 3.1). This is our algorithm's hard case of a root with the maximum val-multiplicity, namely d_1; so we need to branch into a special val-multiplicity $= d_1$ branch/node in the tree. We 'stay' here till all the chains of val-mult$=d_1$ roots are explored and stored in an array (D in Algo.1). This is done in the red subpart of the tree in Fig.1. Next, it branches into the easier degree-reduction cases; which is denoted by the green nodes (enclosed by the left rectangle in Fig.1).

As we go down the tree, we perform lifting computationally; in a way, depth h implies that the polynomial has been lifted h times. This has been schematically portrayed in Figure 1, where the red node is a more complicated transformation than the simple lifting (done in green node).

To summarize, the *degree reduction case* of local root is where effective degree reduces ($v < d_1$ suffices, due to Theorem 2.1); while *val-multiplicity d_1 case* is that when val-mult $v = d_1$.

Degree reduction. The crux of Algorithm 1 lies in our Theorem 2.1 which states that optimistically the effective degree reduces "most of the time". The algorithm ends when we either have exhausted the power of p, denoted by k, or when the effective degree becomes 1. The latter case has been explained in more details in Theorem 2.1, and the result implies that a root always exists once the effective degree has fallen down to 1; which is essentially (p-adic) Hensel's lifting on a *linear* polynomial [Hen18].

Since we have degree reduction, we expect the number of steps (or the tree-depth) to be $O(d_1)$. However, the difficulty arises when the effective degree remains unchanged. Here, in the worst-case, factors of p^{d_1} are divided out of the polynomial in lifting steps, and the depth of the branching tree (Figure 1) can extend to $\Omega(k/d_1)$, which we can not afford as it would lead to iteratively going over $p^{\Omega(k/d_1)}$-many local roots; which is like the complexity of the brute-force algorithm.

Hensel's Lifting. Given a non-singular root $\mathbf{a}$ of polynomial $h(\mathbf{x})$, we can lift it to modulo any p-power (like in Theorem 2.1-(2)), using a variant of p-adic Hensel's lifting [Hen18]. Since $\mathbf{a}$ is a non-singular root, at least one of the first-order derivatives of $h(\mathbf{x})$ will not vanish. Corollary 2.4 implies that the val-multiplicity is then exactly 1, and in the next step of lifting, the effective polynomial will be *linear.* If this linear polynomial is of the form $m_1x_1 + m_2x_2 + m_0$, then (say) we can fix x_2 to any value in $[p]$ and find the corresponding unique value of x_1 to yield a root by simple lifting. For the next p-adic coordinate, after lifting, these m_1, m_2 (coefficients of x_1 resp. x_2) will not change; while m_0 might change (due to the lifting performed in the non-effective part of the polynomial). Thus, from Theorem 2.1-(2), we have the fact that the effective polynomial continues to stay as linear, and we can fix the current-coordinate x_2 to find the corresponding x_1 every time; enabling us to lift to modulo any p-power (for arbitrary fixing of x_2 in this example).

Avoiding $\Omega(k)$ lifting steps. In Theorem 2.1, we show that the effective degree does not reduce *only if* the val-multiplicity of the local root is d_1. So, we consider the branches of degree reduction as discussed in previous paragraphs, but tweak our algorithm carefully in val-mult $= d_1$ case to prevent the depth of the tree (and the number of lifting steps) from blowing-up to $\Omega(k/d_1)$.

We want branches of lesser val-multiplicity ($< d_1$) originating from the branches of val-multiplicity $= d_1$ roots, where the effective degree reduces in each of them. So, we create a process of 'removing' these contiguous val-multiplicity d_1 nodes, instead of looping over all of them; and recursively call the root-finding function on each of the degree-reducing branches arising from this single node/polynomial. This removal-process is guided by something called d_1-*forms* (Lemma 3.1), and will be subdivided into single and multiple val-multiplicity d_1 roots. (1) When *multiple* val-multiplicity roots exist, we show in Lemma 3.2 that the polynomial has a special form (namely d_1-*power*). We traverse these cases in a contiguous way. (2) Next we traverse over cases where val-multiplicity d_1 root is unique. At the end, when we encounter lesser val-multiplicity roots, we recursively call the root-finding algorithm. Overall, we find the lengths of these contiguous traversals, as well as the possibilities of the underlying d_1-powers resp. unique-roots. This is discussed in the latter-half of Section 3, by considering a dynamic basis-change on the variable set $\mathbf{x}$, so that we do not have to iterate over 'too many' local roots. This is done by employing the idea of [BLQ13] to find representative-roots of a univariate polynomial system.

Summarizing this case of val-mult$=d_1$ roots, we showed that the possibilities of contiguous chains is small (i.e. polynomial in k, d), and every lesser val-multiplicity branch appearing from these chains is in a degree reduction case. So, in the tree (Fig.1) an intermediate red node is created that 'jumps' over all the val-mult$=d_1$ cases (Sections 3–4). This bounds the depth of our tree to $2d$.

Stopping condition and representative roots. The algorithm terminates when either a root gets completely specified mod p^k, or when effective degree ≤ 1 (any of its roots can be Hensel lifted all the way to our required power of p), or when no root exists. In the third case, the root-set returned is just the emptyset ϕ, while in the first case it is a singleton.

For the second case, roots will be returned in terms of representative roots (Definition A.3). Eg. when the lifted polynomial is zero modulo p^ℓ, any value in $\mathbb{Z}/p^\ell\mathbb{Z}$ is a root, and thus we return $*_1$ resp. $*_2$ for the coordinates x_1 resp. x_2, which represent the entire $\mathbb{Z}/p^\ell\mathbb{Z}$. The roots returned will be $(*_1, *_2)$, with the number of possibilities being $p^{2\ell}$. This will be termed as our usual *representative root*.

When the effective degree is 1: as we sketched before, we can fix one variable as a local root and find the value of the other variable. In such a fashion, given any value of one coordinate, say x_1, we can find each p-adic coordinate of x_2 one by one. Even if only one variable is present in the linear form, say x_2, the other variable x_1 will still be free, and for any given value of x_1, denoted by $*$, we can find the corresponding values of the local roots of x_2, and thus a root of the polynomial modulo p^ℓ. Let us denote this function for determining x_2 from any value of x_1 by $c(\cdot)$, which simply finds each coordinate of x_2 using Hensel's lifting. Thus, the output can be denoted as $(*, c(*))$. The number of roots in this expression is p^ℓ. (Note: This can contribute more roots to the original f mod p^k, so a more careful calculation is done in Section 5.2.) This type of representative root will be termed as *linear-representative*.

Pseudocode. Based on these ideas, we sketch our Algorithm. Its main procedure is the ROOT-FIND() function in Algorithm 1. It takes as *input*: the polynomial $f_j(x_1, x_2)$ and the number p^{k_j} ($k =: k_0$ initially). The algorithm starts with calling ROOT-FIND($f(x_1, x_2), p^k$). If there are valid roots, it outputs the set of roots $R \subseteq (\mathbb{Z}/p^k\mathbb{Z})^2$, otherwise returns ϕ.

The submodule of REMOVE-d_1-FORM() in Algorithm 2 is a procedure to eliminate intermediate computations where effective degree does not decrease. In a way, it speeds-up the search for roots to higher precision coordinates, by jumping over contiguous cases of roots of val-multiplicity d_1. REMOVE-d_1-FORM() outputs an *array* of: linear transformation which can be used to jump over the val-multiplicity d_1 roots, or linear-representative root which directly becomes part of the output.

Algorithm 1 Root Finding of $f_j(x_1, x_2)$ mod p^{k_j}

```
1: procedure ROOT-FIND($f_j(x_1, x_2), p^{k_j}$)
2:     if $k_j \le 0$ OR $f_j(x_1, x_2) \equiv 0 \bmod p^{k_j}$ then return $(*_1, *_2)$
3:     Define $d_1 := \deg(f_j \bmod p)$, $R := \phi$.
4:     if $d_1 = 1$ then
5:         return linear-representative $(*, c(*))$ or $(c(*), *)$, where $c(\cdot)$ is given by Hensel's Lifting.
6:     for $a_1 \in \{0, p-1\}$ do
7:         for $a_2$ such that $f_j(a_1, a_2) \equiv 0 \bmod p$ and val-mult($\mathbf{a}$)$< d_1$ do
8:             $f_{j+1}(x_1, x_2) := p^{-v} f_j(a_1 + px_1, a_2 + px_2)$, where $v := v_p(f_j(a_1 + px_1, a_2 + px_2))$.
9:             $S :=$ ROOT-FIND($f_{j+1}, p^{k_j - v}$)
10:            $R := R \cup (\mathbf{a} + pS)$
11:    if val-multiplicity$= d_1$ root exists then
12:        $D :=$ REMOVE-$d_1$-FORM($f_j, p^{k_j}$)
13:        for $(r_1 + p^{i_1} L_1, r_2 + p^{i_2} L_2, i_3) \in D$ do
14:            Write $f_j$ in basis $\{L_1, L_2\}$ to get $\tilde{f}_j(L_1, L_2) := f_j(x_1, x_2)$.
15:            Lift it to $\tilde{f}_j(L_1, L_2) := p^{-i_3 d_1} \cdot \tilde{f}_j(r_1 + p^{i_1} L_1, r_2 + p^{i_2} L_2)$.
16:            if $k_j - i_3 d_1 \le 0$ then
17:                The roots will be $(r_1 + p^{i_1} \cdot *_1, r_2 + p^{i_2} \cdot *_2)$ in $(L_1, L_2)$ basis.
18:                Consider the tuple $(r_1 + p^{i_1} \cdot *_1, r_2 + p^{i_2} \cdot *_2)$ and perform the inverse linear
                   transformation from $(L_1, L_2)$ to $(x_1, x_2)$ on this tuple as a whole. Store this
                   representative root (with two independent $*$'s) in a set $S$
19:                $R := R \cup S$
20:            else
21:                For $\tilde{f}_j$ mod $p^{k_j - i_3 d_1}$, find the val-mult $< d_1$ local roots and then recursively find
                   all the roots; as done in Steps 6-10. Let this be given by the set $\tilde{R}$.
```

```
22:             For each root $(\tilde{r}_1, \tilde{r}_2) \in \tilde{R}$ of $\tilde{f}_j \bmod p^{k_j - i_3 d_1}$: consider $(r_1 + p^{i_1}\tilde{r}_1, r_2 + p^{i_2}\tilde{r}_2)$ and
                perform inverse linear transformation from $(L_1, L_2)$ to $(x_1, x_2)$ on them. Store
                these final roots $(\bmod\, p^{k_j})$ in a set $S$.
23:             $R := R \cup S$
24:     return $R$
```

2. Evolution of effective degree during lifting steps

In this section, we analyze the effective degree at each step and look more closely as to when this decreases, or remains the same, by looking at the val-multiplicity of the local root during lifting. The proof idea is to analyze the monomials in terms of x_1 and x_2, and see how they behave after the transformation $(x_1, x_2) \mapsto (a_1 + px_1, a_2 + px_2)$ followed by division by appropriate power of p. This can be summed up by the following theorem.

Theorem 2.1 (Degree reduction). *For a polynomial $f(x_1, x_2) \in (\mathbb{Z}/p^k\mathbb{Z})[x_1, x_2]$, given an $\mathbb{F}_p^2$-root (a_1, a_2) of $f(x_1, x_2)$, let us denote $g(x_1, x_2) := p^{-v} f(a_1 + px_1, a_2 + px_2)$, where $v := v_p(f(a_1 + px_1, a_2 + px_2))$. Let the previous effective degree be $d_1 := \deg(f(x_1, x_2) \bmod p)$ and current effective degree be $d_2 := \deg(g(x_1, x_2) \bmod p)$. Then the following holds:*

1. *If $d_1 > 1$, then $d_2 \le v \le d_1$. (So, $d_2 = d_1$ only if $v = d_1$.)*
2. *If $d_1 = 1$, then $d_2 = 1$.*

Before proving this, let us first see the degree evolution in some concrete examples.

Example 2.2. *Let us see how the effective degree could* reduce. *Consider $f(x_1, x_2) = x_1^2 + x_2^3 \bmod p$. This has degree $d_1 = 3$. Clearly, $(0, 0)$ is its root modulo p. So, apply the transformation $(x_1, x_2) \mapsto (0 + px_1, 0 + px_2)$, to get $g(x_1, x_2) := p^{-2} f(px_1, px_2) = x_1^2 + px_2^3$, which has effective degree $d_2 = 2 = v < d_1$.*

Example 2.3. *Let us see why the effective degree might remain* unchanged *in 'many' steps (which is bad for us). Consider $f(x_1, x_2) = x_1^3 + x_2^3 + p^9(x_1 + x_2 - 1) \bmod p^{10}$. Then, around its root $(0, 0)$, use the translation $(x_1, x_2) \mapsto (0 + px_1, 0 + px_2)$ and division by p^3, to reduce to a simpler $g(x_1, x_2) := x_1^3 + x_2^3 + p^6(-1) \bmod p^7$. We do this two more times to reduce to a simpler $g(x_1, x_2) := x_1^3 + x_2^3 - 1 \bmod p$. In these three steps, the degree $= 3 = v$ did not reduce. However, if $p \neq 3$, then the degree will finally reduce in the fourth step. [If $p = 3$ then proceed with $g := x_1 + x_2 - 1$.]*

Proof of Theorem 2.1. We have two cases.

Case 1: $\mathbf{d_1} > \mathbf{1}$. We have a local root (a_1, a_2) such that $v = v_p(f(a_1 + px_1, a_2 + px_2))$. Using Taylor's expansion (Definition A.1), we can write the polynomial in the form (say over $\mathbb{Z}_p$)

$$f(a_1 + px_1, a_2 + px_2) \;=\; \sum_{\ell=0}^{d} \left(\sum_{|\mathbf{i}|=\ell} \frac{\partial_{\mathbf{x}^{\mathbf{i}}} f(\mathbf{a})}{\mathbf{i}!} \cdot (px_1)^{i_1} (px_2)^{i_2} \right), \tag{3}$$

where $d := \deg(f)$. The terms in Equation 3 need to vanish modulo p^v for all $\ell \le v$. In particular, $p^{v-|\mathbf{i}|} \mid \frac{\partial_{\mathbf{x}^{\mathbf{i}}} f(\mathbf{a})}{\mathbf{i}!}$. Suppose $v > d_1$, then by the above equation $f(a_1 + x_1, a_2 + x_2) \equiv 0 \bmod p$, implying $f(\mathbf{x}) \equiv 0 \bmod p$, which contradicts with $d_1 > 1$. Thus, $v \le d_1$.

However, we do have a term which has valuation exactly v (= val-multiplicity of the local root), and this can be obtained only from monomials where $i_1 + i_2 \le v$ (that too in the effective polynomial part). So, the highest degree term surviving among these (in $g \bmod p$) has degree $d_2 \le v \le d_1$.

Remark: The case of $d_2 = d_1$ implies that $v = d_1$. Thus, $p^{v-|\mathbf{i}|} \mid \frac{\partial_{\mathbf{x}^{\mathbf{i}}} f(\mathbf{a})}{\mathbf{i}!}$ for all orders $|\mathbf{i}| < d_1$; while, some order-d_1 partial-derivative at $\mathbf{a}$ has p-valuation exactly 0.

Case 2: $\mathbf{d_1 = 1}$ (Hensel's Lifting)**.** Write $f(x_1, x_2) =: f_1(x_1, x_2) + p \cdot f_2(x_1, x_2)$. We have the effective polynomial $\deg(f_1(\mathbf{x})) = 1$, and hence it can be written as a linear polynomial $m_1x_1 + m_2x_2 + m_0$. Since (a_1, a_2) is a local root, we transform f to get $m_1(a_1 + px_1) + m_2(a_2 + px_2) + m_0 + p \cdot f_2(a_1 + px_1, a_2 + px_2)$. Dividing by p, and going mod$p$, we get in the next step to another linear polynomial: $m_1x_1 + m_2x_2 + m_0'$. So we end up with $d_2 = 1$. $\square$

Using the proof of Theorem 2.1, we get a corollary on partial derivatives of $f(\mathbf{x})$, which motivates the inclusion of the term 'multiplicity' in our new concept of 'val-multiplicity'.

Corollary 2.4. *Local root* $\mathbf{a}$ *of* $f(\mathbf{x})$ *has val-multiplicity* $\geq v$*, if and only if* $p^{v-|\mathbf{i}|} \mid \frac{\partial_{\mathbf{x}^{\mathbf{i}}} f(\mathbf{a})}{\mathbf{i}!}$*, for all orders* $|\mathbf{i}| < v$*.*

Remark 2.5. *Implicitly, the above proofs of Theorem 2.1 and Corollary 2.4 needed $d < p$ so that the factorials could appear in the denominators (in Equation 3). For smaller p, the same proof works 'syntactically'. Formally, consider the* Hasse derivatives *instead of partial-derivatives.*

Using this theorem, we can get an idea of how the effective degree reduces. If a root $(a_1, a_2) \in \mathbb{F}_p^2$ is such that $f(a_1 + px_1, a_2 + px_2) \not\equiv 0 \bmod p^{d_1}$, then we can keep applying the appropriate transformation $(x_1, x_2) \mapsto (a_1 + px_1, a_2 + px_2)$, until the effective degree reduces to 1. Once this effective degree has reduced to 1, we have a *compact description* of all its roots: as we can arbitrarily fix one variable and uniquely find the p-adic value of the other variable.

However, the problem arises when the root (a_1, a_2) is such that $f(a_1+px_1, a_2+px_2) \equiv 0 \bmod p^{d_1}$. In this case, the degree may not reduce, and we might need to keep lifting to k/d_1 steps. This is computationally infeasible, the search-tree becomes very large, and takes time exponential in k/d_1. We tackle this case in the next section.

3. Structure of f via rank of local roots of val-mult$=d_1$

We need to handle the challenge of our local root $\mathbf{a}$ of f having val-multiplicity $v = d_1$. Here, the effective degree does not reduce in the next step. We first show the structure of such $f(x_1, x_2)$.

Lemma 3.1 (d_1-form at $\mathbf{a}$)**.** *If* $\mathbf{a} \in \mathbb{F}_p^2$ *is a root of* $f(\mathbf{x}) \bmod p$ *such that* $f(a_1 + px_1, a_2 + px_2) \equiv 0 \bmod p^{d_1}$*, where d_1 is the effective degree of f, then* $f(\mathbf{x}) \in \langle x_1 - a_1, x_2 - a_2 \rangle^{d_1}_{\mathbb{F}_p[\mathbf{x}]}$*.*

Proof. Recall Taylor's expansion (Definition A.1) and Corollary 2.4. Write $f(\mathbf{x})$ as $f((x_1 - a_1) + a_1, (x_2 - a_2) + a_2) = \sum_{\ell=0}^{\infty} A_\ell$, where,

$$A_\ell := \sum_{|\mathbf{i}|=\ell} \frac{\partial_{\mathbf{x}^{\mathbf{i}}} f(\mathbf{a})}{\mathbf{i}!} \cdot (x_1 - a_1)^{i_1} (x_2 - a_2)^{i_2} .$$

By Corollary 2.4, we know that the A_t's, for $t < d_1$, vanish modulo p as the root has val-multiplicity $v = d_1$. Furthermore, for $t > d_1$, A_t's vanish modulo p; as f has effective degree d_1 and this A_t has derivatives of order $> d_1$. Thus, all A_t's, apart from A_{d_1}, vanish modulo p. So, the polynomial f is of the form $\sum_i c_i \cdot (x_1 - a_1)^{d_1 - i}(x_2 - a_2)^i$, which is the required d_1*-form in* $\mathbf{x} - \mathbf{a}$. $\square$

In Lemma 3.1's situation, if $\mathbf{a}$ is unique, then using the structure of f we can easily find the root (eg. a simple search in $\mathbb{F}_p^2$), and lift without getting into multiple val-mult$=d_1$ branching. A serious obstruction arises when there are several local roots $\mathbf{a}$ of val-multiplicity $= d_1$. We will now show the extra special structure of such an $f(x_1, x_2)$.

W.l.o.g let $\mathbf{0}$ be a local root of val-multiplicity $= d_1$. This means that $f \in \langle x_1, x_2 \rangle^{d_1}_{\mathbb{F}_p[\mathbf{x}]}$. If another local root $\mathbf{a} \neq \mathbf{0}$ exists with val-multiplicity $= d_1$, then we also have $f(\mathbf{x}) \in \langle x_1 - a_1, x_2 - a_2 \rangle^{d_1}_{\mathbb{F}_p[\mathbf{x}]}$. So, $f \in \langle x_1, x_2 \rangle^{d_1} \cap \langle x_1 - a_1, x_2 - a_2 \rangle^{d_1}$, over $\mathbb{F}_p[\mathbf{x}]$. Then, we show f to be a *perfect-power*!

Lemma 3.2 (Two val-mult=d_1 roots)**.** *For a polynomial $f \in \mathbb{F}_p[x_1, x_2]$ of degree d_1, if $f \in \langle x_1, x_2\rangle^{d_1}_{\mathbb{F}_p[\mathbf{x}]} \cap \langle x_1 - a_1, x_2 - a_2\rangle^{d_1}_{\mathbb{F}_p[\mathbf{x}]}$, for some $\mathbf{a} \neq \mathbf{0} \in \mathbb{F}_p^2$, then we have $f \equiv c(a_2x_1 - a_1x_2)^{d_1} \bmod p$, where $c \in \mathbb{F}_p^*$.*

Proof. W.l.o.g., assume that $a_1 \in \mathbb{F}_p^*$. Thus, we have

$$\begin{aligned}\langle x_1 - a_1, x_2 - a_2\rangle^{d_1} &= \langle x_1 - a_1,\, a_1x_2 - a_1a_2\rangle^{d_1} \\ &= \langle x_1 - a_1,\, a_1x_2 - a_1a_2 - a_2(x_1 - a_1)\rangle^{d_1} \qquad (4)\\ &= \langle x_1 - a_1,\, a_1x_2 - a_2x_1\rangle^{d_1}.\end{aligned}$$

Also, $\langle x_1, x_2\rangle^{d_1} = \langle x_1, a_1x_2 - a_2x_1\rangle^{d_1}$ (as $a_1 \neq 0$). The intersection of these two ideals modulo the ideal $\langle a_1x_2 - a_2x_1\rangle$ is: $\langle a_1x_2 - a_2x_1\rangle + \langle x_1(x_1 - a_1)\rangle^{d_1}$ (as x_1 and $x_1 - a_1$ are coprime mod $a_1x_2 - a_2x_1$). Since f has effective degree less than $2d_1$, we deduce: $(a_2x_1 - a_1x_2) \mid f$.

The quotient $f/(a_2x_1 - a_1x_2) \in \langle x_1, a_2x_1 - a_1x_2\rangle^{d_1-1} \cap \langle x_1 - a_1, a_2x_1 - a_1x_2\rangle^{d_1-1}$. Clearly, degree of this quotient polynomial is $d_1 - 1$. So, we can repeat this process to show that $(a_2x_1 - a_1x_2)^{d_1} | f$; which makes the two equal up to a constant multiple. ☐

Hence, we see that if a polynomial f has two val-mult=d_1 roots with one of them being zero and the other being $\mathbf{a} \neq \mathbf{0}$, then the effective polynomial $f \bmod p$ is of the form $(a_2x_1 - a_1x_2)^{d_1}$. This means that f is d_1-th power of a linear polynomial iff rank of the val-mult=d_1 roots is two (i.e. multiple such roots). In the case of unique val-mult=d_1 root we will call the polynomial *d_1-nonpower-form*, while that for multiple val-mult=d_1 roots, we call the polynomial *d_1-power*.

Branching in d_1-nonpower-form. In this case, find the unique val-multiplicity d_1 root, and do the lifting step. There is no branching required.

Branching in d_1-power. W.l.o.g. the effective polynomial will be of the form $(a_2x_1 - a_1x_2)^{d_1}$. So, there are p roots (of val-mult=d_1): (a_1t, a_2t) for any $t \in \mathbb{F}_p$. This leads to branching, which we will avoid, by taking a different route.

The first observation (Lemma 3.3) is that d_1-nonpower-form can not lead to a d_1-power. Thus, we deduce that whenever a contiguous chain of d_1-power lifting ends, then every d_1-form in the subsequent contiguous lifting steps is a d_1-nonpower-form.

Lemma 3.3 (Nonpower to power?)**.** *If f is a d_1-nonpower-form having a single val-mult=d_1 root $\mathbf{a}$, then its lift $p^{-d_1}f(\mathbf{a} + p\mathbf{x})$ is not a d_1-power.*

Proof. W.l.o.g. we can assume $\mathbf{a} = \mathbf{0}$. Since the effective polynomial is a d_1-form having $(0, 0)$ as the root, it is of the form

$$f(x_1, x_2) \;\equiv\; \sum_{i=0}^{d_1} c_i x_1^i x_2^{d_1-i} \bmod p\,. \qquad (5)$$

After lifting given by $(x_1, x_2) \mapsto (px_1, px_2)$, followed by division by p^{d_1}, this polynomial will become

$$\sum_{i=0}^{d_1} c_i x_1^i x_2^{d_1-i} + g(x_1, x_2)\,,$$

for some polynomial g of degree $\leq d_1 - 1$. Suppose this lift is a d_1-power, say $(L + m_0)^{d_1}$ mod p, where $m_0 \in \mathbb{F}_p$ and L is a linear form in x_1, x_2. Now comparing the degree d_1 homogeneous parts in all these equations, we conclude that $f \equiv L^{d_1}$ mod p. This contradicts the fact that it was a d_1-nonpower-form. Therefore, d_1-nonpower-forms can not become d_1-powers in one lifting step. ☐

So we mainly need to study the case where: a d_1-power, say L^{d_1} is followed by another d_1-power, say L'^{d_1}, in the next lifting step. In the next subsection we unearth the structure that goes in the formation of L' after lifting the polynomial $L^{d_1} + \langle p \rangle$. This will give us the optimized bound on the branching of the red-nodes of the tree.

3.1. **Structure of consecutive d_1-powers.** For a d_1-form, the effective polynomial $f(x_1, x_2)$ mod p will be of the form L^{d_1}, for some linear polynomial L (eg. $x_1 + x_2 + 1$). W.l.o.g. assume $\{L, x_2, 1\}$ to be of rank=3 (over $\mathbb{F}_p$). Let us rewrite f in the basis $\{L, x_2\}$, instead of $\{x_1, x_2\}$, denoted by $\tilde{f}(L, x_2)$ $(= f(\mathbf{x}))$. Since it is an invertible linear transformation, it now suffices to find roots of

$$\tilde{f} \;=:\; L^{d_1} + p \cdot L^{d_1-1} \cdot u_1(x_2) + p \cdot L^{d_1-2} \cdot u_2(x_2) + \cdots + p \cdot u_{d_1}(x_2)\,. \tag{6}$$

Lift d_1-power to d_1-power. Suppose after lifting by $p^{-d_1}\tilde{f}(pL, x_2)$, the effective polynomial is *again* a d_1-power; then it has to be the case that

$$L^{d_1} + L^{d_1-1} \cdot u_1(x_2) + L^{d_1-2} \cdot u_2(x_2)/p + \cdots + u_{d_1}(x_2)/p^{d_1-1} \;\equiv\; (L + u_1(x_2)/d_1)^{d_1} \bmod p\,, \tag{7}$$

for some univariate polynomials u_j's, such that Equation 7 is a perfect power of the linear polynomial $L + u_1(x_2)/d_1$. Consequently, those local roots a_2 for which the above system is satisfied, transform previous L to $p\,(L + u_1(a_2)/d_1)$ in this lifting step.

Considering RHS expansion, we also obtain equations, for $j \in [d_1]$, as

$$u_j(x_2) \;\equiv\; p^{j-1}\binom{d_1}{j} \cdot (u_1(x_2)/d_1)^j \bmod p^j\,. \tag{8}$$

Note: In case $p|d_1$, the above modulus can be further increased to clear away p-multiples from the denominator. In this fashion, we create a system of modular equations (in x_2) for the first step of lifting. Moving on, we consider the next lift.

Two consecutive d_1-power liftings. The effective polynomial after the first step was $(L + c(x_2))^{d_1}$. Let us look at the polynomials obtained before division by p^{d_1}. It was $(pL + pc(x_2))^{d_1} + \langle p^{d_1+1} \rangle$. Composing this with another lift of the same kind, the polynomial has to be of the form $(p(pL+pc(x_2)) + p^2\tilde{c}(x_2))^{d_1} + \langle p^{2d_1+1} \rangle$. This implies that we can as well directly lift $L \mapsto p^2L$, divide by p^{2d_1}, and find the value of $c(x_2) + \tilde{c}(x_2)$. So, Eqn.7 can be replaced by the lift $p^{-2d_1}\tilde{f}(p^2L, x_2)$ equalling a d_1-power:

$$L^{d_1} + L^{d_1-1} \cdot u_1(x_2)/p + L^{d_1-2} \cdot u_2(x_2)/p^3 + \cdots + u_{d_1}(x_2)/p^{2d_1-1} \;\equiv\; (L + u_1(x_2)/pd_1)^{d_1} \bmod p\,, \tag{9}$$

Furthermore, we can write down the univariate modular equations like Eqn.8.

In this way any i-length contiguous chain of d_1-power liftings, can be directly written as a system of univariate modular equations like Eqn.8. It comes from the constraint that the lift $p^{-id_1}\tilde{f}(p^iL, x_2)$ has to equal a d_1-power mod p. Next, this system can be solved by adapting [BLQ13] (see Algorithm 3) to get the representative roots for x_2 variable. Of course, on substituting this in x_2, we will know the final d_1-power L'^{d_1} that the i many lifts yield.

How many consecutive d_1-powers? The length of this chain can be at most k/d_1. So, we go over all $i \le \lfloor k/d_1 \rfloor$. Iterating over them in decreasing order, we find all the possible ways of getting d_1-powers (before moving to other cases). This ensures that we do not miss any $(\mathbb{Z}/p^k\mathbb{Z})$-root of f in the search-tree.

This can be illustrated through the following examples.

Example 3.4. *Consider the polynomial $f(x_1, x_2) = x_1^2 \bmod p^k$. The d_1-power contiguous chain will be of length $k/2$; and each time $L = x_1$. The corresponding root will be $(p^{k/2} \cdot *_1, *_2)$.*

Example 3.5. *We use a slightly more complicated polynomial this time, with $f(x_1, x_2) = (x_1 + x_2)^2 + p(2(x_1+x_2)x_2 + px_2^2) + p^6h(x_1, x_2)$. Here the contiguous chain of d_1-power liftings has length 3. The first linear polynomial is $L := x_1 + x_2$ and the base-change gives $\tilde{f}(L, x_2) = L^2 + p(2Lx_2 + px_2^2) + p^6\tilde{h}(L, x_2)$.*

The polynomial after another lifting will be $(L+x_2)^2 + p^4\tilde{h}(pL, x_2)$. Now, the new linear form will be $L' := L + x_2$. This we can continue lifting, keeping L' unchanged, for 2 more steps.

Notation for x_2 representatives. A problem arises when the representative for x_2 is $*_2$, i.e. x_2 can take any p-adic value. Eg. if we lift $f = L^{d_1} + p^{d_1}x_2^{d_1+1}$ (with free $x_2 = *_2$) then we get $g := L^{d_1} + x_2^{d_1+1}$. The degree of the new polynomial has now *increased*, which we never want to happen in our tree branchings. In order to prevent this, we preprocess the representative root $x_2 = *_2$ by increasing the precision by one coordinate, and thus increasing the number of representative roots by a factor of p (hence, more branchings in the tree). In other words, we consider the representative as $a + p \cdot *_2$, for $a \in \{0, \dots, p-1\}$.

The following lemma shows that the effective degree never grows in lifting steps, in our algorithm.

Lemma 3.6 (Degree invariant)**.** *The effective degree in each transformation described for d_1-forms is always d_1.*

Proof. Let us follow the above notation in the basis $\{L, x_2\}$ and start with effective degree d_1. We now know that every lifting step looks like the map: $L \mapsto pL$ and $x_2 \mapsto a_2 + p^{i_2}x_2$ (for some $i_2 \geq 1$ and integer a_2), followed by division by p^{d_1}. As we calculated in Theorem 2.1, such a lifting step yields effective degree $\leq d_1$. Since we are in the d_1-form case, this implies that the effective degree remains d_1 always. $\square$

Summing up. The structure discovered above gives a natural pseudocode that we describe in Algorithm 2. The contiguous val-mult=d_1 chain will have some d_1-powers, say i_1 many, followed by i_3 many d_1-nonpower forms, from which we have $i_1 + i_3 \leq k/d_1$. The d_1-nonpower forms can not lead to d_1-powers again, due to Lemma 3.3. Also, to get the i_1 many d_1-powers, we need to use Algorithm 3 and get representatives R_1 for x_2 (in general L_2, independent of L_1). Going over each $i_1, i_3 \leq \lfloor k/d_1 \rfloor$, and each of the representatives R_1, we compute the intermediate representative-roots R (and could continue with our recursion on the local roots with subsequent degree-reduction). This algorithm makes sure that we 'jump' the cases of val-mult=d_1= effective-degree quickly and reach the degree-reduction branchings of the tree.

4. Remove-d_1-Form() subroutine: Proof of Theorem 1.1

In Algorithm 1, since we are iterating over all the possible roots ($O(p^2d)$-many), we do not want the lifting to go on for several steps, since the time complexity is exponential in the number of steps (=tree-depth). The favorable case is when the effective-degree reduces, e.g., when the val-multiplicity of local root is $< d_1$ (from Theorem 2.1). As we will see, once we organize all the possible branches in the tree as portrayed in Figure 1, which is a modified branching w.r.t. all possible val-multiplicities, the effective degree will reduce at each level when we go down the tree. This is what the Remove-d_1-Form() subroutine will achieve inside the red nodes. In this section we sketch the pseudocode based on the ideas developed in Section 3.1.

Data structure returned. In order to lift in such a way, we return an array of tuples of the form $(a_1 + p^{u_1}L_1, a_2 + p^{u_2}L_2, u_3)$. This gives us information on jumping over the val-multiplicity d_1 roots by first covering the d_1-powers followed by d_1-nonpowers. This is done in a basis (L_1, L_2) of variables possibly different from (x_1, x_2). As in Equation 9, we form equations in terms of L_2 and find the roots, such that after lifting according to these (representative) roots, the effective polynomial will be $L_1^{d_1}$. Note that in each lifting according to the fixed part of the representative root, the linear polynomial will change by only a constant. Therefore, $\{1, L_1, L_2\}$ will also span the same space as that of $\{1, x_1, x_2\}$. So, given a root in (L_1, L_2) basis, we can recover the root in (x_1, x_2) basis uniquely.

With information from this tuple, we can do the following sequence of liftings in 'one-shot': i_1-steps of d_1-powers, followed by i_3-steps of d_1-nonpower-forms.

Pseudocode. Summing up, REMOVE-d_1-FORM( ) in Algorithm 2 is a subroutine used in ROOT-FIND( ) in order to jump over the intermediate local roots of val-multiplicity d_1 without invoking recursive-branching. After this, we find the local roots of val-multiplicity $< d_1$ and continue recursively to degree reducing cases in ROOT-FIND( ) (Steps 6-10 of Algorithm 1).

The *input* is the polynomial with and the prime-power, while the *output* is a tuple of linear polynomials, denoting intermediate representative-roots (over $(\mathbb{Z}/p^k\mathbb{Z})^2$), and length of the d_1-forms chain covered.

Algorithm 2 Finding consecutive intermediate val-mult=d_1 roots in one-shot

```
 1: procedure REMOVE-$d_1$-FORM($f(x_1, x_2), p^k$)
 2:     Define $d_1 := \deg(f \bmod p)$, $R := \phi$.
 3:     for $i_1 \in \{\lceil k/d_1 \rceil, \ldots, 0\}$ do
 4:         $R_1 := \phi$
 5:         Find the linear polynomial $L$ such that $f \equiv L^{d_1} \bmod p$. If $L$ is $x_2$-multiple, then set
            $L_2 := x_1$, otherwise set $L_2 := x_2$.
 6:         Compute the (basis-change) polynomial $\tilde{f}$ such that $\tilde{f}(L, L_2) = f(x_1, x_2)$.
 7:         Write $\tilde{f}$ as in Equation 9 and form (univariate, modular) equations like Equation 8 in
            terms of polynomials in $L_2$ such that $i_1$-many contiguous $d_1$-powers exist (Section 3.1).
 8:         Find the representative-roots, in $\mathbb{Z}/p^{i_1 d_1}\mathbb{Z}$, of the system of equations formed in terms
            of $L_2$ (as in the previous step) using Algorithm 3 and store them into $R_1$.
 9:         for each representative-root $r_2 + p^{i_2}* \in R_1$ do
10:             Find linear polynomial $L_1$ obtained in the end, by substituting the representative in
                $L_2$, using the method of Section 3.1. Note: $i_2 \geq 1$ and $L_1$ has to be of the form $L + c$
                for some integer $c$.
11:             Write $\tilde{f}(L, L_2)$ in basis $L_1, L_2$ given by $g(L_1, L_2) := \tilde{f}(L, L_2)$.
12:             Lift $g(L_1, L_2) := p^{-i_1 d_1} \cdot g(p^{i_1} L_1 \, , \, r_2 + p^{i_2} L_2)$
13:             for $i_3 \in \{ \lceil k/d_1 \rceil - i_1, \ldots, 0\}$ do
14:                 if $\exists \mathbf{r}' \in (\mathbb{Z}/p^{i_3}\mathbb{Z})^2$ s.t. $g$ is a $d_1$-nonpower-form consecutively $i_3$-times then
15:                     In each precision $\mathbf{r}'$ is unique; so it can be searched easily in the space $\mathbb{F}_p^2$.
16:                     $R_0 := (\, p^{i_1} r'_1 + p^{i_1+i_3} L_1 \, , r_2 + p^{i_2} r'_2 + p^{i_2+i_3} L_2 \, , i_1 + i_3 \,)$
17:                     if $R_0 \notin R$ then
18:                         $R := R \cup \{R_0\}$
19:                 else
20:                     break
21:     return R
```

4.1. **Proof of correctness of Algorithm 1.** We prove the correctness of our root finding algorithm (Algorithms 1 and 2) in the following theorem.

Theorem 4.1 (Correctness of Algorithm 1). *Given a polynomial $f(x_1, x_2) \in \mathbb{Z}[x_1, x_2]$ of degree d, a prime p and an integer k. Algorithm 1 using Algorithm 2 as a subroutine, correctly returns all the roots $(a_1, a_2) \in (\mathbb{Z}/p^k\mathbb{Z})^2$ of $f(x_1, x_2) \bmod p^k$, in deterministic $\mathsf{poly}((k + d + p)^d)$ time.*

Proof. In this proof, we will analyze the structure of the roots-tree given by Figure 1. As described in Sections 1.4 and 4, all the green nodes in this tree are in degree-reducing cases since we are calling the recursion to the next step using ROOT-FIND( ) only on val-multiplicity $< d_1$ roots (Steps 9 & 21 of Algorithm 1).

Thus, whenever we go down this tree by iterating over all the roots of val-multiplicity $< d_1$, effective degree d_1 decreases by at least 1. Here, we fix one variable to any value in $\{0, \dots, p-1\}$, and find the p-many possibilities of the other variable. The red node of Figure 1 also gives val-multiplicity $< d_1$ roots after two levels in the tree. Thus, the depth of this tree is at most $2d$, after which we reach the *leaves*. At a leaf, either the precision of p^k has been reached *fixing* a root, or the effective polynomial has become *linear*, or *no* root exists.

For the width/fanin of the tree, the number of possible lengths of d_1-powers is $\lceil k/d_1 \rceil = O(k)$, while that of d_1-nonpower-forms is also $O(k)$. Therefore, the number of possibilities of lengths of val-mult=d_1 branches is $O(k^2)$. In Algorithm 2, all the val-mult=d_1 branches of length $i_1 + i_3$ for i_1-length contiguous d_1-powers and i_3-length contiguous d_1-nonpower-forms are returned. As we have seen in Section 3.1, we can solve for the intermediate values of L_2, from which we find the appropriate L_1 too, such that d_1-powers are possible. After this, we come straight down to the d_1-nonpower-forms. Furthermore, from Theorem A.4 and Equation 8, the number of representatives for L_2 is $O(d)$, which may become $O(pd)$ after preprocessing to handle $*$.

This bounds the number of roots returned by REMOVE-d_1-FORM(f, p^k) by $O(k^2dp)$. In a deterministic version of Algorithm 3, we exhaustively search for every L_2 root at each step, so its complexity is $\mathsf{poly}(kdp)$-time. Therefore, all the operations inside REMOVE-d_1-FORM() can be performed in $\mathsf{poly}(kdp)$-time.

Since the depth of the tree is $2d$ (in Fig.1), the total size of the tree gets bounded by $\mathsf{poly}((k+d+p)^d)$. Since all other operators are usual field operations and search in $\mathbb{F}_p$, the net time complexity of our algorithms is just a polynomial overhead of going over all these nodes. This proves the complexity to be deterministic $\mathsf{poly}((k+d+p)^d)$-time. ☐

Proof of Theorem 1.1. Algorithm 1, using Algorithm 2 as a subroutine, correctly returns all the roots of $f(x_1, x_2) \bmod p^k$, in a representatives form, in deterministic $\mathsf{poly}((k+d+p)^d)$ time (due to Theorem 4.1)

By the time the algorithm terminates, we will have $O((k^2dp)^d)$ leaves of the tree, where we have either found roots or encountered a dead-end. If roots exist, then each such leaf signifies that either

(1) we reached p^k and a fixed root (a_1, a_2) is returned,
(2) or, we reached p^k and (some invertible linear transformation of) the representative $(p^{i_1} \cdot *_1, p^{i_2} \cdot *_2)$ is returned. The total number of roots in this case is $p^{2k-i_1-i_2}$,
(3) or, when the effective polynomial becomes linear and $(*, c(*))$ is returned (or some invertible linear transformation of it); where $*$ represents all values in $\mathbb{Z}/p^{k-i}\mathbb{Z}$. Here, we fix only one variable to find the value of the other at each step for $k-i$ Hensel lifts, implying that the number of roots is p^{k-i}.

Using this technique to sum over all the leaves which do not lead to dead-ends, we can count the total number of roots in deterministic $\mathsf{poly}((k+d+p)^d)$ time. ☐

5. Other results: Proofs overview

5.1. Computing p-adic roots: Proof of Corollary 1.2.
In this subsection, we give a bound for k_0 in terms of the degree d and the maximum absolute value M of the coefficients, such that finding a root modulo p^{k_0} would imply finding all representative $\mathbb{Z}_p$-roots of f.

Preprocessing– Reduce to radical case. We are concerned with the roots of $f(x_1, x_2)$ in $\mathbb{Z}_p$, which is an integral domain (with fraction field $\mathbb{Q}_p$). The polynomial will have a unique factorization in $\mathbb{Z}_p$, which will be of the form

$$f(x_1, x_2) \;=\; \prod_{i=0}^{r} g_i(x_1, x_2)^{e_i} \;, \tag{10}$$

where $g_i(x_1, x_2)$'s are coprime over $\mathbb{Z}_p$. Even if f has some square-full factors (some e_i's are ≥ 2), we can eliminate them efficiently, by computing its gcd with the first-order derivatives. This will result in the new polynomial being of the form $\prod_{i=0}^{r} g_i(x_1, x_2)$, which we will call the *radical polynomial* $\mathrm{rad}(f)$. The polynomial f and its radical $\mathrm{rad}(f)$ have the same set of roots in $\mathbb{Z}_p$. In the following lemma, we bound the absolute-value of the coefficients of the radical, by M^d.

Lemma 5.1 (Bound for p-adic radical)**.** *If a polynomial f of degree d has the absolute-value of its coefficients bounded by M, then its radical has its coefficients bounded by $M^{O(d)}$.*

Proof. We prove this using Euclid's algorithm for finding gcd, when we consider the gcd of f and any one of its first-order derivative f'.

At each step of Euclid's gcd algorithm, we have two polynomials q_i and q_{i+1}, where $\deg(q_i) \geq \deg(q_{i+1})$. We compute the remainder of q_i when divided by q_{i+1}, assume it to be q_{i+2}, and then proceed to do the same with q_{i+1} and q_{i+2}.

Now, it can be inductively shown that the coefficients of q_i is bounded by M^{F_i}, where F_i is the i-th Fibonacci number. This is true as while dividing q_{i-2} by q_{i-1}, the quotient will have its coefficients bounded by that of q_{i-2}. This quotient multiplied by q_{i-1} will give the bound for the remainder, which thus is bounded by the product of bounds of coefficients of q_{i-1} and q_{i-2}. Now, this procedure continues for $\log d$ steps, implying that the coefficients of the gcd of f and its derivative is bounded by $M^{F_{\log d}} = M^{O(d)}$. Dividing f by this gcd will give the bounds on the coefficient of $\mathrm{rad}(f)$, which is also $M^{O(d)}$. □

Therefore, w.l.o.g. we consider $f(x_1, x_2)$ to be square-free having absolute value of coefficients $\leq M^{O(d)}$, and continue with our algorithm of finding roots in $\mathbb{Z}_p$.

Representative roots in $\mathbb{Z}/p^k\mathbb{Z}$ vs roots in $\mathbb{Z}_p$. The return value of the algorithm, in the base-case, is either the representative root $(*_1, *_2)$ when the exponent of p required gets achieved (Step 2 of Algorithm 1), or linear-representative root $(*, c(*))$ (Steps 4-5 of Algorithm 1).

For large enough k, i.e. $k > k_0$, we want to show that if a representative root $(*_1, *_2)$ is returned, then the fixed part of the root is already a $\mathbb{Z}_p$-root. In the other case, for linear-representative roots, we can simply use Hensel lifting to lift to $\mathbb{Z}_p$-roots (or, to as much precision as we wish).

Discriminant. Let $f'(x_1, x_2)$ be some first-order derivative of $f(x_1, x_2)$. The resultant [CLO13, Chapter 3] of f and f' w.r.t. x_2 is denoted $R(x_1) := \mathsf{Res}_{x_2}(f(x_1, x_2), f'(x_1, x_2))$, which is also one of the *discriminants* of f. $R(x_1)$ is not identically zero in $\mathbb{Z}_p$, as this would imply: f and its derivative have a common factor; contradicting the radical condition.

The roots of $R(x_1)$ given by $\hat{x}_1$ satisfy the condition that the univariate $f(\hat{x}_1, x_2)$ is square-full. Furthermore, given $\hat{x}_1$, we can easily find the values of x_2 (d-many), as it becomes the univariate root-finding problem over $\mathbb{Z}_p$ which has a famous solution (see Algorithm 3).

Bound to distinguish $\mathbb{Z}_p$ roots. The main idea is to find a bound for the exponent of p such that each root returned using root-finding is either a linear-representative root, or a unique lift of this root is a $\mathbb{Z}_p$ root. A similar bound was achieved for univariate polynomials by [DS20]. However, the complications of lifting multivariate roots did not arise there, as every p-adic root corresponded to a p-adic factor.

Lemma 5.2 (Discriminant of radical)**.** *Let $f(x_1, x_2) \in \mathbb{Z}[x_1, x_2]$ be a polynomial of degree d whose coefficients have absolute value bounded above by M. Let its radical polynomial be $g := \mathrm{rad}(f)$. The roots of $R(x_1) = \mathsf{Res}_{x_2}(g, g')$ in $\mathbb{Z}_p$ are in one-one correspondence to the representative roots of $R(x_1) \bmod p^k$, for any $k \geq k_1 := \Theta(d^6 \log M)$.*

Proof. We have the polynomial $f(x_1, x_2)$ of degree d. Its radical polynomial $g := \mathrm{rad}(f)$, has degree $\leq d$ and coefficients bounded by $M^{O(d)}$ (Lemma 5.1).

The resultant polynomial $R(x_1) = \mathsf{Res}_{x_2}(g, g')$ is the determinant of a $(2d+1) \times (2d+1)$ matrix consisting of elements formed from the coefficients of g. Thus, the degree of $R(x_1)$ is $< 2d^2 + d$, and the absolute-value of the coefficients is $< (dM^d)^{2d+1}$.

Now, we need to find a bound on k_1 such that the roots of $R(x_1)$ are in one-one correspondence to those of $g(x_1, x_2) \bmod p^{k_1}$. [DS20, Theorem 20] showed that the representative roots of a univariate polynomial modulo p^k, for $k > d'(\Delta+1)$, are in one-one correspondence to the roots of that polynomial in $\mathbb{Z}_p$, where d' is the degree and Δ is the p-valuation of its discriminant. In our case of finding $\mathbb{Z}_p$-roots of $R(x_1)$, the degree is $2d^2 + d$, while the discriminant is at most $((dM^d)^{2d+1})^{2(2d^2+d)+1}$. Thus, the valuation of discriminant of R is bounded by $O(d^4 \log_p M)$. Substituting the values of d' and Δ, we have $k_1 := \Theta(d^6 \log_p M)$. □

$\mathbb{Z}_p$-roots. Consider $g = \mathrm{rad}(f)$ and $k_1 = \Theta(d^6 \log M)$. Define $g_2(x_2) := \mathrm{Res}_{x_1}(g(x_1, x_2)\,, R(x_1))$. Intuitively, roots x_2 of g_2 come from the roots x_1 of R. So, again applying [DS20, Theorem 20] on this univariate polynomial g_2, it suffices to compute its roots mod p^{k_2}, to compute its distinct p-adic roots; where k_2 is asymptotically $\log_p(p^{k_1 \cdot 2d^2 \cdot d}) = O(d^9 \log M)$.

Using the value of k_2 as above, we find roots of $g(x_1, x_2)$ from Root-Find(g, p^{k_2}). Let $(\tilde{a}_1, \tilde{a}_2)$ be the fixed-part of a root thus obtained. If $R(\tilde{a}_1) \equiv 0 \bmod p^{k_2}$, then the above argument, that defined k_2, ensures that $(\tilde{a}_1, \tilde{a}_2)$ does lift to a $\mathbb{Z}_p$-root of R, g_2, g and f (in this case uniquely).

Non-root of discriminant. Thus, the case left is: $R(\tilde{a}_1) \not\equiv 0 \bmod p^{k_2}$. Consider the univariate $g(\tilde{a}_1, x_2)$. We know that its $\mathbb{Z}_p$-roots are different $\bmod\, p^{k_2}$ and at most d many; one of which is $\tilde{a}_2$. Consider $g_1(x_2) := p^{-v} \cdot g(\tilde{a}_1\,, \tilde{a}_2 + x_2)$, where $v \geq 0$ is the p-valuation of $g(\tilde{a}_1\,, \tilde{a}_2 + x_2)$ as a polynomial over $\mathbb{Z}_p$. Note that x_2 divides g_1, but x_2^2 does not divide g_1 (mod p). Thus, 0 is a simple-root of g_1 and we can potentially Hensel lift it to p-adics.

To implement this formally, we need to increase the precision so that the extra p-factors can be removed from g. Note that if we assume $p \nmid g(\tilde{a}_1, x_2)$ then $v \leq k_2 + (k_2 - 1)(d-1) < d \cdot k_2$. Fix $k_0 := d \cdot k_2 = \Theta(d^{10} \log M)$. Now consider $\tilde{g}(\mathbf{x}) := p^{-v} \cdot g(\tilde{a}_1 + p^{k_2} x_1\,, \tilde{a}_2 + p^{k_2} x_2) \bmod p^{k_0}$. By the argument above, $\tilde{g} \bmod p$ is linear in x_2 (it is easier to see by substituting $x_1 = 0$). Thus, an extension of this root has to end up in some leaf of Root-Find(g, p^{k_0}) algorithm as say $(\tilde{a}_1', \tilde{a}_2')$; which will Hensel lift to p-adic integral root(s) due to the linear x_2 term in the lift.

Since the set of p-adic roots for f and g is the same, we could as well run Root-Find(f, p^{k_0}). This proves the following lemma.

Lemma 5.3 (p^{k_0} is p-adic)**.** *Given a polynomial $f(x_1, x_2) \in \mathbb{Z}[x_1, x_2]$ of degree d and having absolute-value of coefficients bounded by M. Each root represented in the leaves of the tree of* Root-Find(f, p^{k_0})*, for $k_0 := \Theta(d^{10} \log M)$, lifts to a $\mathbb{Z}_p$-root of $f(x_1, x_2)$.*

We further need the condition that the structure of this tree does not change with k for $k \geq k_0$. In order to show that, we prove the following lemma. Denote $R_1(x_1) := \mathsf{Res}_{x_2}(g, \partial_{x_2}(g))$ and $R_2(x_2) := \mathsf{Res}_{x_1}(g, \partial_{x_1}(g))$

Lemma 5.4 (Fix p-adic tree)**.** *If a leaf of the tree given by Lemma 5.3 returns a representative root with the fixed part (a_1, a_2), that is not linear-representative, then $R_1(a_1) = R_2(a_2) = 0 \bmod p^k$. Moreover, (a_1, a_2) lifts to a unique root of f over $\mathbb{Z}_p$; and their number does not change as k grows beyond k_0.*

Also, the tree (Fig.1) in our algorithm does not change, and remains isomorphic, for $k \geq k_0$; except the leaf with the root $\mathbf{0}$.

Proof. As argued above, the representative roots which are not linear-representatives, must satisfy the condition $R_1(a_1) \equiv 0 \bmod p^{k_0}$. Using a similar technique we can show that $R_2(a_2) \equiv 0 \bmod p^{k_0}$ as well.

Assume that for some large enough k, $k \geq k_0$, a new leaf in the tree of Figure 1 appears, with the root (r_1, r_2) such that $R_1(r_1) \equiv R_2(r_2) \equiv 0 \bmod p^k$. However, this leads to a contradiction as the branch corresponding to (r_1, r_2) should have already been present in the tree at precision k_0 in the representative root.

As argued before, this root r_j of $R_j \bmod p^{k_0}$ always lead to $\mathbb{Z}_p$ roots of R_j for $j = 1, 2$ (due to [DS20, Theorem 20]), and that of g_2, g, f. Thus, the number of representative roots can not decrease. Therefore, the number of representative roots which are not linear is fixed once we reach k_0, and hence (a_1, a_2) has a unique lift to $\mathbb{Z}_p$.

Together with Hensel lifting, it is then clear that, the linear-representative roots can neither increase in number, nor reduce, as $k \geq k_0$ grows.

The only change that could happen is, for $k \geq k_0$, if the leaf with fixed root $\mathbf{0}$ is used to lift to $f(p^v x_1, p^v x_2) \bmod p^k$, with $k > v \geq k_0$. This may create a new subtree under the old leaf $\mathbf{0}$; as this type of branches are the only ones that were not explored in Algorithm 1 mod p^{k_0}. $\square$

The following examples should help illustrate the p-adic machinery more clearly.

Example 5.5. *Consider the polynomial* $f = (x_1 - 1)(x_2 - 2) \bmod p^k$. *The first step of our algorithm has to be* $x_1 = 1$ *or* $x_2 = 2$. *Considering the root* $\mathbf{a} := (1, 3)$, *the polynomial after lifting becomes* $x_1(1 + px_2)$, *which is an (effective) linear form; thus, a linear-representative root will be returned, which has* x_2 *as the free variable while* x_1 *will stay fixed to* 1. *This gives the leaf* $\mathbf{r} := (1 + p\mu(*), 3 + p*)$, *and a computable* $\mathbb{Z}_p$*-function* $\mu(\cdot)$, *which allows the p-adic lift of* $\mathbf{a}$. *In this case,* $\mu = 0$.

Example 5.6. *Now, consider* $f(x_1, x_2) = (x_1 - px_2)(x_1 - 2px_2) \bmod p^k$. *Lifting the root* $\mathbf{a} := (0, 1)$ *gives us* $(x_1 - 1 - px_2)(x_1 - 2 - 2px_2)$, *which is not yet effective linear. Choosing the next lifting-step around the root* $(1, 0)$, *the polynomial after lifting becomes* $(x_1 - px_2)(-1 + px_1 - 2p^2 x_2)$, *which is an (effective) linear polynomial; thus, a linear-representative root will be returned, corresponding to* $(x_1 - px_2)$, *which has* x_2 *as the free variable while* x_1 *depends on it. This gives the leaf* $\mathbf{r} := (p + p^3\mu(*),\, 1 + p^2*)$, *and a computable* $\mathbb{Z}_p$*-function* $\mu(\cdot)$, *which allows the p-adic lift of* $\mathbf{a}$. *In this case,* $\mu(w) := w$.

However there are several roots in $\mathbb{Z}_p$, which can not be noticed modulo p^{k_0}, because they are indistinguishable from $\mathbf{0}$. This is seen in the following example.

Example 5.7. *Consider the polynomial* $x_1^3 + x_2^3 \bmod p^k$, *for* $p > 3$ *and* $3|k$. *Some of its linear-representative roots are* $(p^j + p^{j+1}*\,, -p^j + p^{j+1}\mu(*))$, *for any* $j < k/3$ *and* $\mu(w) := -w$. *Also,* $(p^{k/3}*_1\,, p^{k/3}*_2)$ *is a non linear-representative root. It can lift to the p-adic root* $\mathbf{0}$, *but it can also lift to* $(p^k, -p^k)$; *which our algorithm could not explore due to the precision being only* p^k.

The following theorem completes the connection, of Algorithm 1, with *all* p-adic roots of f. Basically, it scales up the roots by p^v-multiple, whenever possible, and creates a new data-structure for representatives in the leaves of the fixed tree modulo mod p^{k_0}, in Fig.1. It can also be seen as a way of further *blowing-up* the leaf of the fixed tree that gives the $\mathbf{0}$ root.

Theorem 5.8 (High val p-adic roots)**.** *We can efficiently 'expand' our leaves as follows:*

(1) Define a set of representative-roots $\mathcal{H}_v$, *for* $v \geq k_0$, *s.t. for each root* $\mathbf{a} \in \mathcal{H}_v$, $p^v\mathbf{a}$ *lifts to a p-adic root of* f.

(2) We can compute the fixed tree for $\mathcal{H}_{k_0}$ *efficiently by Algorithm 1. The other sets* $\mathcal{H}_v$, *for* $v > k_0$, *lift from the same representatives as in the leaves of* $\mathcal{H}_{k_0}$; *so we do not recompute them.*

Let (r'_1, r'_2) *be a p-adic root of* f. *Then,* $\exists v \geq 0$, $\exists$ root $\mathbf{a} \in \mathcal{H}_v$ *lifting to* $\mathbf{a}'$, *for which* $(r'_1, r'_2) = p^v\mathbf{a}'$. *In this sense, our fixed tree covers all p-adic roots of* f.

Proof. Let u be the p-valuation of $\mathbf{r}'$, i.e. $p^u || (r'_1, r'_2)$. If $u = \infty$, i.e. $(r'_1, r'_2) = \mathbf{0}$, then clearly some leaf in the set $\mathcal{H}_{k_0}$ will satisfy the required statement.

If $u < k_0$, then $\mathbf{r}' \neq 0 \bmod p^{k_0}$; so it will be covered in some nonzero leaf of the tree of Lemma 5.4.

Assume $\infty > u \geq k_0$. Now consider the system $f(p^u \cdot \mathbf{x}) = 0$, say over $\mathbb{Z}_p$. Write $f =: \sum_{m \leq i \leq d} f_i$ into homogeneous-parts, with m being the least-degree part ($f_m \neq 0$). Thus, by homogeneity, the system becomes

$$0 \;=\; f(p^u \cdot \mathbf{x})/p^{um} \;=\; \sum_{m \leq i \leq d} p^{u(i-m)} \cdot f_i(x_1, x_2)\,. \tag{11}$$

If $f = f_m$ (i.e. f is homogeneous), then $f(p^u \cdot \mathbf{x}) = 0$ iff $f(\mathbf{x}) = 0$. Thus, $\mathcal{H}_v$, for all $v \geq 0$, is given by the fixed tree in Lemma 5.4 and we are done.

Assume $f \neq f_m$ (i.e. f is *in*homogeneous). Then, the above system implies: $f_m(x_1, x_2) \equiv 0 \bmod p^u$. Since f_m has bounded coefficients and $u \geq k_0$, we compute the fixed tree (Lemma 5.4 for $f_m \bmod p^{k_0}$) efficiently; and all its leaves (except $\mathbf{0}$) lift to p-adic roots. Each leaf, in our Algorithm 1 can be viewed as defining a nontrivial p-adic map $\mu : \mathbb{Z}_p \to \mathbb{Z}_p$ s.t. w.l.o.g., $f_m(w, \mu(w)) = 0$, where w is a variable. Check whether $f(p^u w\,, p^u \mu(w)) = 0$. Then, by repeating this argument (on f_{m+1} etc.) we can deduce: $f_i(w, \mu(w)) = 0$ for all $m \leq i \leq d$. Since, μ is a p-adic function common to these polynomials, that are all upper bounded by the parameters of f, we can learn μ by working with each f_i just $\bmod\, p^{k_0}$.

Algorithmically, we find this common μ by first invoking Algorithm 1 on $f_m \bmod p^{k_0}$ and then verifying it for $f(p^{k_0} w\,, p^{k_0} \mu(w)) \equiv 0 \bmod p^{k_0(d+1)}$. [Or, we could construct the tree common to the system $\{f_m, \ldots, f_d\} \bmod p^{k_0}$.]

But this shows an interesting property p-adically that $f(p^{k_0} w\,, p^{k_0} \mu(w)) = 0$ iff $f_i(w, \mu(w)) = 0$, for all $m \leq i \leq d$ iff $f(p^u w\,, p^u \mu(w)) = 0$, for all $u \geq 0$. Essentially, the high-valuation roots arise only from homogeneous polynomial system! □

Lemmas 5.3-5.4 and Theorem 5.8 describe the p-adic nature of the tree and the representative roots, after the threshold bound of k_0. This finishes the proof of Corollary 1.2.

5.2. **Computing Igusa's local zeta function: Proof of Corollary 1.3.** We will show how to compute the Poincaré series, by expressing the number of roots of $f(x_1, x_2) \bmod p^k$, for every k, in a special form. In this subsection, we sketch this algorithm for bivariates using Section 5.1. With more work it can be generalized to n-variates based on the machinery of Section 5.3; thus proving the rationality of the Poincaré series in general.

When we consider f modulo p^k, for large enough k's, the fixed-part of the representative-roots will correspond to p-adic roots, while the remaining-part has 'free' coordinates, eg. $(*_1, *_2)$, which get fixed as we increase k. For $k = k_0$, denote $\mathcal{R}$ as the subset of representative-roots which are not linear-representative roots, while the set $\mathcal{L}$ as the set of linear-representative roots.

Recall the bound of $k_0 = \Theta(d^{10} \log M)$ (Lemma 5.3) to distinguish between $\mathbb{Z}/p^k\mathbb{Z}$ and $\mathbb{Z}_p$-roots: For small values of k, i.e. $k < k_0$, we can count the number of roots in deterministic time $\mathsf{poly}((d + p + \log M)^d)$. For large k's, however, we want to prove a special form to sum up the infinite Poincaré series.

Non linear-representative roots $\mathcal{R}$. Consider a root in $\mathcal{R}$; its fixed-part, say $\mathbf{r}$, will lift to $\mathbb{Z}_p$-roots of f. Its representative part appears due to the contribution of extra p-powers by the other derivatives that appear in the Taylor-series around $\mathbf{r}$. Let e be the multiplicity of $\mathbf{r}$: which can be found as the (largest) e such that $f \in \langle \mathbf{x} - \mathbf{r} \rangle^e$, but $f \notin \langle \mathbf{x} - \mathbf{r} \rangle^{e+1}$.

Now, f can be written as

$$f \;=\; \sum_{i=0}^{e} c_i(x_1, x_2) \cdot (x_1 - r_1)^i (x_2 - r_2)^{e-i}\,. \tag{12}$$

Define $v := v_p(\gcd(c_i(\mathbf{r}) \mid i))$. Since $\mathbf{r}$ is not a common root of c_i's, this value (e, v) does not change as we increase k and make $\mathbf{r}$ more precise. Consider the $\mathbb{Z}_p$-root $\mathbf{r}'$ that $\mathbf{r}$ lifts to. Let us now consider the ways in which the digits of $\mathbf{r}'$ may be perturbed, and yet it be a root (mod p^k), as we increase k arbitrarily. Let ℓ_1 denote the length up to which we want to keep r_1 equal to r_1' and the rest to $*$ (similarly define ℓ_2). Consider

$$(13) \qquad f(r_1' + p^{\ell_1}x_1 \,, r_2' + p^{\ell_2}x_2) \;=\; \sum_{i=0}^{e} c_i(\mathbf{r}' + p^{\mathbf{l}}\mathbf{x}) \cdot (r_1' - r_1 + p^{\ell_1}x_1)^i \cdot (r_2' - r_2 + p^{\ell_2}x_2)^{e-i} \,.$$

The valuation of this expression is the minimum of $v_p(c_i(\mathbf{r})) + \ell_1 i + \ell_2(e-i)$, over all i. If this is $\geq k$ then $\mathbf{x}$ can take any value in $\mathbb{Z}/p^{k-\ell_1}\mathbb{Z} \times \mathbb{Z}/p^{k-\ell_2}\mathbb{Z}$, and extend, by the above equation, to a root mod p^k. This is what should happen as we are not in the linear-representative case. We need to count these possibilities, which will give us the number of ways the root $\mathbf{r}$ could lift as k increases. For this, we should consider only those possibilities of (ℓ_1, ℓ_2) that are *minimal*, i.e. $(\ell_1 - 1, \ell_2)$ and $(\ell_1, \ell_2 - 1)$ should violate the linear-inequality system. This is a system of half-spaces in the plane, forming an open polygon $\mathcal{P}$ with either $\leq e$ vertices or just one hyperplane. It can be checked that in all cases the counting function

$$(14) \qquad \sum_{(\ell_1, \ell_2) \in \mathcal{P}} p^{(k-\ell_1)} \cdot p^{(k-\ell_2)}$$

can be rewritten as a sum of $p^{u_i(k)}$, $i \leq e$, where $u_i(k)$ is a linear function in k over $\mathbb{Q}$. Let us call this sum $N_{k,\mathbf{r}}(f)$. Importantly, the number of summands here is fixed, and does not grow with k.

The following example illustrates the notion of this polytope.

Example 5.9. *Consider the polynomial $f(x_1, x_2) = x_1^2 x_2 \bmod p^k$. Here, for the root $(0,1)$, the value of e is 2, and accordingly ℓ_1, the precision of x_1 required, is $= k/2$. However, the value of e for $(1,0)$ is 1 and $\ell_2 = k$. For the root $\mathbf{0}$, both variables contribute powers of p, where they are zero with precision ℓ_1, ℓ_2 respectively. Then, we must have $2\ell_1 + \ell_2 \geq k$, which gives the hyperplane in ℓ_1, ℓ_2; summing over all these values we can calculate Eqn.14.*

Linear-representative roots $\mathcal{L}$. Consider a linear-representative root $\mathbf{r} \in \mathcal{L}$, with fixed part as $\mathbf{a}$ of length (e_1, e_2) respectively. Up to linear transformations, we can claim that our algorithm defines a computable $\mathbb{Z}_p$-function $\mu(\cdot)$ s.t. for all $u \in \mathbb{Z}_p$, $f(x_1 + a_1 + p^{e_1}u \,, x_2 + a_2 + p^{e_2}\mu(u)) \;=\; 0$. Using this fact, we write Equation 12 in the ideal form, defining (e, v) as the largest integers s.t., in $\mathbb{Z}_p[\mathbf{x}]$,

$$(15) \qquad f(x_1 + a_1 \,, x_2 + a_2 + p^{e_2}\mu(0)) \;\in\; p^v \cdot \langle x_1 \,, x_2 \rangle^e \;+\; \langle x_1 \,, x_2 \rangle^{e+1} \,.$$

Note that we have defined (e, v) by fixing $u = 0$. The motivation is that if we use some other u in $\mathbb{Z}_p$, we will get the same values (e, v). Otherwise, say for some $0 \neq u \in \mathbb{Z}_p$,

$$f(x_1 + a_1 + p^{e_1}u \,, x_2 + a_2 + p^{e_2}\mu(u)) \;\in\; p^{v'} \cdot \langle x_1 \,, x_2 \rangle^{e'} \;+\; \langle x_1 \,, x_2 \rangle^{e'+1} \,.$$

Then, by Lemma 5.4, $e' = e$, because the tree remains isomorphic, even when we make the root more precise than $k \geq k_0$. In the same algorithm, lifting steps cause division by p-powers and reach the leaf $\mathbf{r}$, so v' remains v even when we make the root more precise than $k \geq k_0$.

Fix $u \in \mathbb{Z}/p^{k-e_1}\mathbb{Z}$, and consider the unique p-adic root $\mathbf{r}'$ that the leaf $\mathbf{r}$ gives above. We can now follow the counting process done after Equation 13. Varying u, the polytope boundary $\mathcal{P}$ does not change (similar to the argument given after Equation 15); on the other hand, fixing $(\ell_1 - e_1)$-digits of u (resp. ℓ_1 value), fixes that many in $\mu(u)$ (resp. ℓ_2 value). Thus, we get a *partial* count (slightly different from Equation 14) as:

$$\sum_{(\ell_1,\ell_2)\,\in\,\mathcal{P}\,\cup\,\{(k-e_2)-(\ell_1-e_1)\,\leq\,(k-\ell_2)\}} p^{(\ell_1-e_1)}\cdot p^{(k-e_1)-(\ell_1-e_1)}\cdot p^{(k-e_2)-(\ell_1-e_1)}\;. \tag{16}$$

We call this sum $N'_{k,\mathbf{r}}(f)$; and is written as a sum of $p^{u_i(k)}$, $i \leq e$, where $u_i(k)$ is a *linear* function in k over $\mathbb{Q}$.

Blowing-up root 0. What is missing here are the roots with valuation in the interval $[k_0, k-1]$, because these are zero mod p^{k_0} and so they get missed in $\mathcal{L}$ and $\mathcal{R}$. To account for these, we need to resort to the set $\mathcal{H}_{k_0}$, constructed in Theorem 5.8 that 'blows-up' the leaf node $\mathbf{0}$ in the tree of finding roots modulo p^{k_0}.

Now, from the way Algorithm 1 works, the representatives in $\mathcal{H}_{k_0}$ generate a disjoint set of $(\mathbb{Z}/p^k\mathbb{Z})$-roots, which are in number $= \sum_{\mathbf{r}\in\mathcal{H}_{k_0}} N_{k_0,\mathbf{r}}$. This sum is easy to precompute (by Theorem 1.1), as it is independent of k. Each of these roots can be multiplied by p^e, for $e \in [k_0 \ldots k-1]$, to get the 'high'-valuation roots. Thus, the total number of such roots is $= (k-k_0)\cdot\sum_{\mathbf{r}\in\mathcal{H}_{k_0}} N_{k_0,\mathbf{r}}$.

Overall, the above summands account for all the $\mathbb{Z}/p^k\mathbb{Z}$-roots of f, for $k \geq k_0$.

Summing up, the number of roots modulo p^k, for $k < k_0$, can be counted by Theorem 1.1. Fixing $k = k_0$, as described above, we compute the data related to the fixed tree; which has the linear-representative roots in $\mathcal{L}$, $\mathcal{H}_{k_0}$, and the remaining representative-roots in $\mathcal{R}$.

Then, the number of roots of f modulo p^k, $N_k(f)$, is given by

$$N_k(f) \;=\; \sum_{\mathbf{r}\in\mathcal{R}} N_{k,\mathbf{r}}(f) \;+\; \sum_{\mathbf{r}\in\mathcal{L}} N'_{k,\mathbf{r}}(f) \;+\; (k-k_0)\cdot \sum_{\mathbf{r}\in\mathcal{H}_{k_0}} N_{k_0,\mathbf{r}}\;. \tag{17}$$

Recall that the number of roots modulo p^k due to any representative root $\mathbf{r}$ (or a leaf in the fixed tree in Theorem 5.8) is $N_{k,\mathbf{r}}(f)$ resp. $N'_{k,\mathbf{r}}(f)$; defined separately in Equations 14 and 16.

Now, the Poincaré series is given by (eg. see [DS20])

$$P(t) \;=\; P_0(t) \,+\, \sum_{\mathbf{r}\in\mathcal{R}} P_{\mathbf{r}}(t) \,+\, \sum_{\mathbf{r}\in\mathcal{L}} Q_{\mathbf{r}}(t) \,+\, \left(\sum_{\mathbf{r}\in\mathcal{H}_{k_0}} N_{k_0,\mathbf{r}}\right)\cdot \sum_{k\geq k_0} (k-k_0)\cdot (t/p)^k\;, \tag{18}$$

where $P_0(t) = \sum_{k<k_0} N_k(f)\cdot(t/p)^k$, $P_{\mathbf{r}}(t) := \sum_{k\geq k_0} N_{k,\mathbf{r}}(f)\cdot(t/p)^k$ and $Q_{\mathbf{r}}(t) := \sum_{k\geq k_0} N'_{k,\mathbf{r}}(f)\cdot(t/p)^k$.

The expression of Equation 17 is a sum of either $p^{u_i(k)}$ or $u_i(k)$, $i \leq O(d)$, where $u_i(k)$ is a *linear* function in k over $\mathbb{Q}$. Thus, Equation 18 can be easily expressed in a 'closed-form formula' by summing the geometric progression over k's, as was done in [DS20, Lemma 23]. Thus, $P_{\mathbf{r}}(t), Q_{\mathbf{r}}(t)$ are rational functions in $\mathbb{Q}(t)$. Therefore, the rational function for the Poincaré series $P(t)$ is computable as promised in Corollary 1.3.

5.3. Generalization to n-variates: Proof of Theorem 1.4.

In this subsection, we generalize our approach to root-finding and counting to n-variate polynomial $f(\mathbf{x})$ using similar techniques as used in bivariates, and reducing the problem to finding roots of $(n-1)$-variate polynomial systems. In order to do so, we first show a modification to Algorithm 1 in order to solve a system of polynomial equations $\bmod\, p^k$. This subsection gives an overview of how to extend the single bivariate root-finding algorithm to that for solving a *system* of bivariates. Then, using that how to solve 3-variate systems. The proof can be straightforwardly generalized to n-variates, for any $n \geq 3$.

Solving bivariates simultaneously. Suppose we have m polynomials $f_1(x_1,x_2),\ldots,f_m(x_1,x_2)$, each of degree $\leq d$. At each step of lifting, we iterate over all the roots of each polynomial separately, but in parallel, such that the local roots of each iteration are common to every polynomial.

This can be intuitively thought of as creating trees like Figure 1 corresponding to each polynomial, whose nodes are 'isomorphic', i.e. we branch corresponding to a root if and only if the root is present in the trees of all the polynomials. Also, whatever linear transformation we apply on $\mathbf{x}$ acts on all the f_i's simultaneously. Thus, we refer to this process as *parallel* search-tree.

We continue growing these trees (and considering only those branches which correspond to a common root); with effective degree decreasing as we go down, until the stopping conditions are obtained. When some polynomial has a representative root $(*_1, *_2)$, then we can proceed to finding roots of the remaining set of polynomial. However, when linear-representative roots are obtained for some polynomial, the analysis can be divided into two cases by their *rank*. When the linear forms (i.e. given by coefficients of x_1 and x_2) are of rank 1 or 2. For rank 2, we can check for a unique root by solving simple linear equations; so, they either have one common root, or none.

The difficulty arises when these linear forms are of rank 1. Again, the isomorphism of the trees corresponding to each polynomial will be used, but after reducing this problem to solving simultaneous equations over lesser number of polynomials.

Solving bivariates– rank= 1 linear forms. Suppose we have the polynomials in the form $ax_1 + bx_2 + c_i + ph_i(x_1, x_2)$, where $a, b, c_i \in \{0, \ldots, p-1\}$, for all $i \in [m]$. If all the c_i's are not the same, we obviously do not have a solution; so we terminate this branch. Assume $c_i = c$ and write the polynomials in the basis of $L := (ax_1 + bx_2 + c)$ and x_2, w.l.o.g assuming $a \neq 0$. (Otherwise, we can use the basis L, x_1). We have the system as

$$(19) \qquad L \equiv pg_1(L, x_2) \bmod p^k \,;\; L \equiv pg_2(L, x_2) \bmod p^k \,; \qquad \ldots \,;\; L \equiv pg_m(L, x_2) \bmod p^k \,,$$

where g_i's can be obtained from h_i's by using the change of basis. So, we get the local-root here, namely L must be $0 \bmod p$ in the current step (based on any value of x_2). Hence we lift L to pL. This grows the tree further. Performing this transformation and subtracting the first equation from each of the other equations, we have :

$$(20) \quad L \equiv g_1(pL, x_2) \bmod p^{k-1} \;;\; 0 \equiv \tilde{g}_2(pL, x_2) \bmod p^{k-1} \,; \qquad \ldots \,;\; 0 \equiv \tilde{g}_m(pL, x_2) \bmod p^{k-1} \,,$$

where $\tilde{g}_i = g_i(L, x_2) - g_1(L, x_2)$, for $i \in \{2, \ldots, m\}$.

Now, on the $(m-1)$ $\tilde{g}_i$, we apply another instance of Algorithm 1, in a parallel way. For a fixing of x_2 from the latter $m-1$ equations in Equation 20, we uniquely get the value of L from the first equation (where the effective polynomial is linear in L). Using these values, we lift both L and x_2, creating a Figure 1-like tree where the branches are such that they satisfy all the m equations. Finally, at the leaf we get the representatives for x_2 too, and halt the algorithm.

Thus, we create the m isomorphic trees for $O(d)$ many lifting steps as before (Figure 1), then restrict the linear condition on the first polynomial and simultaneously solve the next $m-1$ polynomials modulo a smaller power of p and continue with our construction of $m-1$ isomorphic trees. Using this subroutine, we can find all the roots of the system of bivariate equations in time complexity same as that of Theorem 1.1, along with a multiplicative overhead of m for storing this array of polynomials in each node.

Solving 3-variate— Lifting Step. Given a root $\mathbf{a} \in \mathbb{F}_p^3$ and polynomial $f_j(\mathbf{x})$ in the j-th step, we find the polynomial after lifting as $f_{j+1}(\mathbf{x}) = p^{-v} f_j(\mathbf{a} + p\mathbf{x})$, where $v = v_p(f_j(\mathbf{a} + p\mathbf{x}))$ is the val-multiplicity of the root $\mathbf{a}$. Theorem 2.1 can be similarly proved to show that effective degree reduces in all cases other than when $d_1 = 1$ or $v = d_1$. We again form a tree similar to Figure 1 with the invariant that effective degree must reduce along depth.

Solving 3-variate— Val-multiplicity d_1 case. As in Algorithm 2, we again 'jump' over the val-multiplicity d_1 cases directly so that the recursion can continue to degree reduction cases. Again, Lemma 3.1 can be proved for 3-variates to show that the polynomial must have the d_1-form $\langle x_1 - a_1, x_2 - a_2, x_3 - a_3 \rangle^{d_1}$, where $\mathbf{a}$ is a val-multiplicity d_1 root. However, when multiple val-multiplicity d_1 roots exist, w.l.o.g. given by $\mathbf{0}$ and $\mathbf{a}$, where $a_1 \neq 0$, we can modify the proof of

Lemma 3.2 to show that the effective polynomial is zero modulo $\langle a_1x_2 - a_2x_1, a_1x_3 - a_3x_1\rangle^{d_1}$. The rank of the nonzero roots, given by $\langle \mathbf{x} - \mathbf{a}\rangle$, can be 1 or 2 (in general, $1, 2, \ldots, n-1$ for n-variates).

In the case of rank=2 val-mult=d_1 roots, there will be only one linear polynomial, say $f \in \langle a_1x_2 - a_2x_1\rangle^{d_1} + \langle p\rangle$. So, the equations will be like Eqn.7, except that the polynomials u_j's will be in two variables, say x_2 and x_3. We will form a bivariate system, and solve it using the simultaneous bivariate root-finding as discussed above; to find all the representative-roots for x_2, x_3. This new version of Algorithm 2 will take $\mathsf{poly}((k + d + p)^d)$-time; and make the tree this much wider.

In the case of rank=1 val-mult=d_1 roots, we will have a multinomial expansion having two linear forms $\{a_1x_2 - a_2x_1, a_1x_3 - a_3x_1\}$ instead of a single binomial expansion as done in Equation 9. So, now, we have two linear forms L, L' instead of a single L_1. We need to find the values of the third variable, say x_3, such that the resulting effective polynomial after lifting is again in $\langle L, L'\rangle^{d_1}$. This gives constraints similar to Equation 8. From these, we can use the univariate Algorithm 3 to solve them.

Finally, the techniques of Section 3.1 can be smoothly generalized to search for the contiguous d_1-forms in $\mathsf{poly}((k + d + p)^d)$-time. In particular, we will search them in the decreasing order of the rank of underlying val-mult=d_1 roots: rank=2, rank=1 and then rank=0 in the last.

Conclusion. Using these techniques, we can find the roots of $f(\mathbf{x}) \bmod p^k$, for 3-variates where the only difference from bivariates is the handling of contiguous val-multiplicity d_1 roots (due to the more possibilities of d_1-forms). The same extension can be performed for n-variate m polynomials, for any constant n. Thus, Algorithm 1 fits well in the general framework, and finds all the roots.

Time complexity. For a 3-variate polynomial f, at each step of the tree (Fig.1), there are $O(dp^3)$ branches corresponding to val-mult$< d_1$ roots. For val-mult=d_1 roots, there are three ordered possibilities in the chain: rank=2 val-mult=d_1 root, rank=1 val-mult=d_1 roots, and rank=0 (unique) val-mult=d_1 roots. Similar to Lemma 3.3, we can show that rank can not increase after lifting by a val-mult=d_1 root.

For rank=2, we solve a system of bivariate equations. The number of possible branches of bivariates is $O((k^2d + p^2)^{2d})$. Furthermore, the number of possibilities of these special contiguous chains is $O(k^3)$. Therefore, the total number of leaves of the tree for 3-variates will be $O(\,((k^2d + p^2)^{2d} \cdot k^3 + p^3)^d\,)$, which is bounded by $O((k + d + p)^{(4d)^2})$. Assume, for induction hypothesis, that the time complexity, for any n, is bounded by $O((k + d + p)^{(2d(n-1))^{n-1}})$.

For n-variates ($n \geq 3$), the val-multiplicity d_1 roots will have ranks from $n-1$ to 0. For rank=$(n-1)$, we get the maximum upper bound. We will be solving a system of $(n-1)$-variate equations, which will lead to $O((k + d + p)^{(2d(n-2))^{n-2}})$ possibilities and representative-roots. Furthermore, there are k^{n-1} possibilities for the chain. Thus, one tree size is $s_1 := O(\,(k^{n-1}(k+d+p)^{(2d(n-2))^{n-2}} + p^n)^{2d}\,)$. But we may need to repeat this $n-1$ times on each leaf, when we have a system of n-variates to solve. Thus, the time becomes $s_1^{n-1} = O((k + d + p)^{(2d(n-1))^{n-1}})$.

Using this technique for finding all the roots modulo p^k for n-variates, we can generalize the algorithms for finding $\mathbb{Z}_p$-roots and computing the Igusa local zeta function for a system of m polynomials in n-variables as well. For finding roots in $\mathbb{Z}_p$, we consider the resultant w.r.t. one variable at a time, find a bound similar to Lemma 5.2, and proceed with the analysis of roots which are in $\mathbb{Z}_p$ or are not roots of the discriminant. At each step, the bounds according to one variable at a time will be obtained, which get multiplied to give a bound k_0 for $f(\mathbf{x})$ such that roots of $f(\mathbf{x}) \bmod p^{k_0}$ gives us roots which correspond to $\mathbb{Z}_p$ roots as well. This will be a generalization of Theorem 5.8 to n-variates. Similarly we can count roots, where the number of possibilities due to linear-representative roots depends on the rank of the linear forms, and the sum will again be a rational form.

The complexity of finding $\mathbb{Z}_p$ points and that of computing Igusa local zeta function will remain deterministic $\mathsf{poly}((m \log M + p + d)^{(2d(n-1))^{n-1}})$-time.

6. Future work

Root finding of bivariates and of n-variates for any constant n gives rise to the following questions of finding faster algorithms.

(1) Can root-finding be performed for constant d and n, in polylog(p)-time?
(2) Can root-finding of $f(x_1, \ldots, x_n)$ be reduced to root-finding of $f(x_1, \mathbf{a}t + \mathbf{b})$, with high probability, for $\mathbf{a}, \mathbf{b} \in (\mathbb{Z}/p^k\mathbb{Z})^{n-1}$? This would also handle the general n case.

Acknowledgements. We thank Ashish Dwivedi for useful discussions on many related problems in this area. N.S. thanks the funding support from DST-SERB (CRG/2020/000045) and N.Rama Rao Chair.

References

[Apo98] Tom M Apostol, *Introduction to analytic number theory*, Springer Science & Business Media, New York, NY, USA, 1998. 3

[Apo10] ———, *Zeta and related functions.* 3

[Ber67] Elwyn R Berlekamp, *Factoring polynomials over finite fields*, Bell System Technical Journal **46** (1967), no. 8, 1853–1859. 1

[Ber70] ———, *Factoring polynomials over large finite fields*, Mathematics of computation **24** (1970), no. 111, 713–735. 1

[BGGI09] Paula Bustillo, Domingo Gómez, Jaime Gutierrez, and Alvar Ibeas, *A strategy for recovering roots of bivariate polynomials modulo a prime.*, IACR Cryptol. ePrint Arch. **2009** (2009), 233. 3

[BKW19] Andreas Björklund, Petteri Kaski, and Ryan Williams, *Solving systems of polynomial equations over gf (2) by a parity-counting self-reduction*, 46th International Colloquium on Automata, Languages, and Programming (ICALP), 2019, Patras, Greece, LIPIcs, vol. 132, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 26:1–26:13. 4

[BLQ13] Jérémy Berthomieu, Grégoire Lecerf, and Guillaume Quintin, *Polynomial root finding over local rings and application to error correcting codes*, Applicable Algebra in Engineering, Communication and Computing **24** (2013), no. 6, 413–443. 2, 3, 4, 5, 7, 12, 28

[BM67] BJ Birch and K McCann, *A criterion for the p-adic solubility of diophantine equations*, The Quarterly Journal of Mathematics **18** (1967), no. 1, 59–63. 4

[Bre86] Richard P. Brent, *Some integer factorization algorithms using elliptic curves*, Australian Computer Science Communications **8** (1986), 149–163. 2

[BW10] Maheshanand Bhaintwal and Siri Krishan Wasan, *Generalized reed–muller codes over $\mathbb{Z}_q$*, Designs, Codes and Cryptography **54** (2010), no. 2, 149–166. 3

[CG00] David G. Cantor and Daniel M. Gordon, *Factoring polynominals over* p*-adic fields*, Algorithmic Number Theory, 4th International Symposium, ANTS-IV, Leiden, Netherlands, Lecture Notes in Computer Science, vol. 1838, Springer, 2000, pp. 185–208. 2

[CGRW19] Qi Cheng, Shuhong Gao, J Maurice Rojas, and Daqing Wan, *Counting roots for polynomials modulo prime powers*, The Open Book Series **2** (2019), no. 1, 191–205. 2

[Chi87] Alexander Leonidovich Chistov, *Efficient factorization of polynomials over local fields*, Doklady Akademii Nauk **293** (1987), no. 5, 1073–1077. 2

[Chi21] Alexander L Chistov, *An effective algorithm for deciding solvability of a system of polynomial equations over p-adic integers*, Algebra i Analiz **33** (2021), no. 6, 162–196. 4, 5

[CHJK$^+$94] AR Calderbank, AR Hammons Jr, P Vijay Kumar, NJA Sloane, and P Solé, *The z4-linearity of kerdock, preparata, goethals and related codes*, IEEE Trans. Inf. Theory **40** (1994), no. 2, 301–319. 3

[CL01] Howard Cheng and George Labahn, *Computing all factorizations in $z_n[x]$*, Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation (ISSAC), Ontario, Canada, ACM, 2001, pp. 64–71. 2

[CLO13] David Cox, John Little, and Donal O'Shea, *Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra*, Springer Science & Business Media, 2013. 16

[Con03] J Brian Conrey, *The riemann hypothesis*, Notices of the AMS **50** (2003), no. 3, 341–353. 3

[Cop96a] Don Coppersmith, *Finding a small root of a bivariate integer equation; factoring with high bits known*, International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 1996, pp. 178–189. 3

[Cop96b] ______, *Finding a small root of a univariate modular equation*, International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 1996, pp. 155–165. 3

[Cop97] ______, *Small solutions to polynomial equations, and low exponent rsa vulnerabilities*, Journal of cryptology **10** (1997), no. 4, 233–260. 3

[Cor04] Jean-Sébastien Coron, *Finding small roots of bivariate integer polynomial equations revisited*, International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2004, pp. 492–505. 3

[CRW20] Qi Cheng, J Maurice Rojas, and Daqing Wan, *Computing zeta functions of large polynomial systems over finite fields*, arXiv preprint arXiv:2007.13214 (2020). 3

[CZ81] David G Cantor and Hans Zassenhaus, *A new algorithm for factoring polynomials over finite fields*, Mathematics of Computation **36** (1981), no. 154, 587–592. 1, 28

[Den84] Jan Denef, *The rationality of the poincaré series associated to the p-adic points on a variety*, Invent. math **77** (1984), no. 1, 1–23. 3, 4

[DH01] Jan Denef and Kathleen Hoornaert, *Newton polyhedra and igusa's local zeta function*, Journal of number Theory **89** (2001), no. 1, 31–64. 3

[DM97] Bruce Dearden and Jerry Metzger, *Roots of polynomials modulo prime powers*, European Journal of Combinatorics **18** (1997), no. 6, 601–606. 2

[DMS19] Ashish Dwivedi, Rajat Mittal, and Nitin Saxena, *Counting Basic-Irreducible Factors Mod p^k in Deterministic Poly-Time and p-Adic Applications*, Proceedings of 34th Computational Complexity Conference (CCC 2019), Springer, 2019, pp. 15:1–15:29. 2, 3, 5

[DMS21] Ashish Dwivedi, Rajat Mittal, and Nitin Saxena, *Efficiently factoring polynomials modulo p^4*, Journal of Symbolic Computation **104** (2021), 805–823. 2, 3, 5, 28

[DPR61] Martin Davis, Hilary Putnam, and Julia Robinson, *The decision problem for exponential diophantine equations*, Annals of Mathematics (1961), 425–436. 2

[DS20] Ashish Dwivedi and Nitin Saxena, *Computing igusa's local zeta function of univariates in deterministic polynomial-time*, Open Book Series **4** (2020), no. 1, 197–214. 2, 3, 4, 16, 17, 18, 21

[EK90] Andrzej Ehrenfeucht and Marek Karpinski, *The computational complexity of (xor, and)-counting problems*, International Computer Science Inst., 1990. 2, 5

[Gau00] Pierrick Gaudry, *An algorithm for solving the discrete log problem on hyperelliptic curves*, International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2000, pp. 19–34. 2

[GCM21] Aditya Gulati, Sayak Chakrabarti, and Rajat Mittal, *On algorithms to find p-ordering*, Conference on Algorithms and Discrete Applied Mathematics, Springer, 2021, pp. 333–345. 2, 28

[GGL08] Parikshit Gopalan, Venkatesan Guruswami, and Richard J Lipton, *Algorithms for modular counting of roots of multivariate polynomials*, Algorithmica **50** (2008), no. 4, 479–496. 2, 3, 5

[GH00] Pierrick Gaudry and Robert Harley, *Counting points on hyperelliptic curves over finite fields*, International Algorithmic Number Theory Symposium, Springer, 2000, pp. 313–332. 2

[GNP12] Jordi Guàrdia, Enric Nart, and Sebastian Pauli, *Single-factor lifting and factorization of polynomials over local fields*, Journal of Symbolic Computation **47** (2012), no. 11, 1318–1346. 2

[Gou97] Fernando Q Gouvêa, *p-adic numbers*, p-adic Numbers, Springer, 1997. 27

[GS98] Venkatesan Guruswami and Madhu Sudan, *Improved decoding of reed-solomon and algebraic-geometric codes*, Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280), IEEE, 1998, pp. 28–37. 3

[Har15] David Harvey, *Computing zeta functions of arithmetic schemes*, Proceedings of the London Mathematical Society **111** (2015), no. 6, 1379–1401. 3

[Hen18] Kurt Hensel, *Eine neue theorie der algebraischen zahlen*, Mathematische Zeitschrift **2** (1918), no. 3, 433–452. 2, 7

[HW99] M-D Huang and Y-C Wong, *Solvability of systems of polynomial congruences modulo a large prime*, computational complexity **8** (1999), no. 3, 227–257. 4, 5

[Igu74] Jun-ichi Igusa, *Complex powers and asymptotic expansions. i. functions of certain types.*, Journal für die reine und angewandte Mathematik (1974), 110–130. 4

[Igu77] Jun-Ichi Igusa, *Some observations on higher degree characters*, American Journal of Mathematics **99** (1977), no. 2, 393–417. 4

[Igu07] Jun-ichi Igusa, *An introduction to the theory of local zeta functions*, vol. 14, American Mathematical Soc., 2007. 3

[Kal82] Erich Kaltofen, *A polynomial-time reduction from bivariate to univariate integral polynomial factorization*, 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), IEEE, 1982, pp. 57–64. 2

[Kal85] ______, *Polynomial-time reductions from multivariate to bi-and univariate integral polynomial factorization*, SIAM Journal on Computing **14** (1985), no. 2, 469–489. 2

[Kay05] Neeraj Kayal, *Solvability of a system of bivariate polynomial equations over a finite field*, International Colloquium on Automata, Languages, and Programming, Springer, 2005, pp. 551–562. 4

[Ked01] Kiran S Kedlaya, *Counting points on hyperelliptic curves using monsky-washnitzer cohomology*, Journal of the Ramanujan Mathematical Society (2001), 323–338. 2

[Ked04] ______, *Computing zeta functions via p-adic cohomology*, International Algorithmic Number Theory Symposium, Springer, 2004, pp. 1–17. 3

[Kob12] Neal Koblitz, *p-adic numbers, p-adic analysis, and zeta-functions*, vol. 58, Springer Science & Business Media, 2012. 27

[KRRZ20] Leann Kopp, Natalie Randall, Joseph Rojas, and Yuyu Zhu, *Randomized polynomial-time root counting in prime power rings*, Mathematics of Computation **89** (2020), no. 321, 373–385. 2

[KU11] Kiran S Kedlaya and Christopher Umans, *Fast polynomial factorization and modular composition*, SIAM Journal on Computing **40** (2011), no. 6, 1767–1802. 1

[Lau06] Alan GB Lauder, *A recursive method for computing zeta functions of varieties*, LMS Journal of Computation and Mathematics **9** (2006), 222–269. 3

[LJ87] Hendrik W Lenstra Jr, *Factoring integers with elliptic curves*, Annals of mathematics **126** (1987), no. 3, 649–673. 2

[LL03] Reynald Lercier and David Lubicz, *Counting points on elliptic curves over finite fields of small characteristic in quasi quadratic time*, International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2003, pp. 360–373. 2

[LMMS94] Frank Lehmann, Markus Maurer, Volker Müller, and Victor Shoup, *Counting the number of points on elliptic curves over finite fields of characteristic greater than three*, International Algorithmic Number Theory Symposium, Springer, 1994, pp. 60–70. 2

[LN94] Rudolf Lidl and Harald Niederreiter, *Introduction to finite fields and their applications*, Cambridge university press, 1994. 3

[LPT$^+$17] Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, Ryan Williams, and Huacheng Yu, *Beating brute force for systems of polynomial equations over finite fields*, Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2017, pp. 2190–2202. 4, 5

[Mat70] Ju V Matijasevic, *Enumerable sets are diophantine*, Soviet Math. Dokl., vol. 11, 1970, pp. 354–358. 2

[Mau01] Davesh Maulik, *Root sets of polynomials modulo prime powers*, Journal of Combinatorial Theory, Series A **93** (2001), no. 1, 125–140. 2

[McD74] Bernard R McDonald, *Finite rings with identity*, vol. 28, Marcel Dekker Incorporated, 1974. 3

[MCT02] Kazuto Matsuo, Jinhui Chao, and Shigeo Tsujii, *An improved baby step giant step algorithm for point counting of hyperelliptic curves over finite fields*, International Algorithmic Number Theory Symposium, Springer, 2002, pp. 461–474. 2

[Mil85] Victor S Miller, *Use of elliptic curves in cryptography*, Conference on the theory and application of cryptographic techniques, Springer, 1985, pp. 417–426. 2

[MVZ93] Alfred J Menezes, Scott A Vanstone, and Robert J Zuccherato, *Counting points on elliptic curves over $\mathbb{F}_{2^m}$*, Mathematics of computation **60** (1993), no. 201, 407–420. 2

[NRS17] Vincent Neiger, Johan Rosenkilde, and Éric Schost, *Fast computation of the roots of polynomials over the ring of power series*, Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation, 2017, pp. 349–356. 4, 5

[NZM91] Ivan Niven, Herbert S Zuckerman, and Hugh L Montgomery, *An introduction to the theory of numbers*, John Wiley & Sons, 1991. 3

[Pan95] Peter N Panayi, *Computation of Leopoldt's P-adic regulator.*, Ph.D. thesis, University of East Anglia, Norwich, England, 1995. 2, 4, 28

[Rie59] Bernhard Riemann, *Ueber die anzahl der primzahlen unter einer gegebenen grosse*, Ges. Math. Werke und Wissenschaftlicher Nachlaß **2** (1859), no. 145-155, 2. 3

[Rón87] Lajos Rónyai, *Factoring polynomials over finite fields*, 28th Annual Symposium on Foundations of Computer Science (sfcs 1987), IEEE, 1987, pp. 132–137. 1

[RRZ21] Caleb Robelle, J Maurice Rojas, and Yuyu Zhu, *Sub-linear point counting for variable separated curves over prime power rings*, arXiv preprint arXiv:2102.01626 (2021), 18. 2, 4

[Săl05] Ana Sălăgean, *Factoring polynomials over z4 and over certain galois rings*, Finite fields and their applications **11** (2005), no. 1, 56–70. 2

[Sat02] Takakazu Satoh, *On p-adic point counting algorithms for elliptic curves over finite fields*, International Algorithmic Number Theory Symposium, Springer, 2002, pp. 43–66. 2

[Sch95] René Schoof, *Counting points on elliptic curves over finite fields*, Journal de théorie des nombres de Bordeaux **7** (1995), no. 1, 219–254. 2

[Sir17] Carlo Sircana, *Factorization of polynomials over z/(pn)*, Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation, 2017, pp. 405–412. 2

[Sud96] Madhu Sudan, *Maximum likelihood decoding of reed solomon codes*, Proceedings of 37th Conference on Foundations of Computer Science, IEEE, 1996, pp. 164–172. 3

[Sud97] ———, *Decoding of reed solomon codes beyond the error-correction bound*, Journal of complexity **13** (1997), no. 1, 180–193. 3

[THB86] Edward Charles Titchmarsh and David Rodney Heath-Brown, *The theory of the riemann zeta-function*, Oxford university press, 1986. 3

[Zas69] Hans Zassenhaus, *On hensel factorization, i*, Journal of Number Theory **1** (1969), no. 3, 291–311. 2

[Zas78] ———, *A remark on the hensel factorization method*, Mathematics of Computation **32** (1978), no. 141, 287–292. 2

[ZG03] WA Zuniga-Galindo, *Computing igusa's local zeta functions of univariate polynomials, and linear feedback shift registers*, Journal of Integer Sequences **6** (2003), 36. 2, 3, 5

Appendix A. Preliminaries

We describe some useful notation and previous works.

We use $\mathbf{x}$ to denote the tuple $(x_1, x_2, \ldots, x_n)$. Operations are similarly defined as $\mathbf{a}+\mathbf{b} := (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n)$, and $c \cdot \mathbf{a} := (c \cdot a_1, \ldots, c \cdot a_n)$, for a scalar c. Similarly, for $\mathbf{i} = (i_1, i_2, \ldots, i_n)$, we have $\mathbf{x}^{\mathbf{i}} = x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}$ with degree $|\mathbf{i}|$, and $\mathbf{i}! := i_1!i_2! \ldots i_n!$.

Based on the Taylor's expansion of polynomials in univariates, we define multivariate Taylor's expansion.

Definition A.1 (Taylor's expansion/ series)**.** *Given a polynomial $f(\mathbf{x})$ of degree d, we can write it as (over any characteristic)*

$$f(\mathbf{a}+\mathbf{x}) \;=\; \sum_{\ell=0}^{\infty} \left(\sum_{|\mathbf{i}|=\ell} \frac{\partial_{\mathbf{x}^{\mathbf{i}}} f(\mathbf{a})}{\mathbf{i}!} \cdot \prod_{j=1}^{n} x_j^{i_j} \right), \tag{21}$$

where $\partial_{\mathbf{x}^{\mathbf{i}}} f := \frac{\partial^{i_1+\cdots+i_n} f}{\partial x_1^{i_1} \ldots \partial x_n^{i_n}}$ is an order $|\mathbf{i}|$ *partial derivative.*

For a prime p, we can write any integer a as a power series $a =: a_0 + a_1 p + a_2 p^2 + \ldots$, for $a_i \in \{0, 1, \ldots, p-1\}$. We write $\tilde{a} \in \mathbb{Z}_p$, *the ring of p-adic integers*, as a tuple $(a_0, a_1, a_2, \ldots)$. The j-th coordinate corresponds to a_j, and $\tilde{a} \bmod p^k$ is defined as the projection upto the $(k-1)$-th coordinate, i.e. $a_0 + a_1 p + \ldots a_{k-1} p^{k-1}$. Similarly, we define *the field of p-adic numbers* as the fraction field of $\mathbb{Z}_p$, denoted as $\mathbb{Q}_p$. For more literature on p-adic numbers, we direct the reader to [Gou97, Kob12].

Definition A.2 (Valuation)**.** *For an integer n and a prime p, we define its* valuation *w.r.t. p, denoted $v_p(n)$, as the largest integer v such that $p^v | n$.*

Definition A.3. *A representative of a ring R, denoted by the symbol $*$, takes all values in the ring R. Formally, it is the set $* := \{a | a \in R\}$.*

We further define the operations:

- $b + * = \{b + a | a \in *\}$ for $b \in R$,
- $b* = \{ba | a \in *\}$ for $b \in R$.

Using this definition, for $\beta + p^\ell * \subseteq \mathbb{Z}/p^k\mathbb{Z}$ for $\beta \in \mathbb{Z}/p^\ell\mathbb{Z}$, $\ell \le k$, we have

$$\beta + p^\ell * = \{\beta + p^\ell a | a \in \mathbb{Z}/p^{k-\ell}\mathbb{Z}\} \tag{22}$$

In a similar fashion, a *representative root* of a polynomial $f(x) \in \mathbb{Z}/p^k\mathbb{Z}$ is denoted by a set $\beta + p^\ell *$ for $\beta \in \mathbb{Z}/p^\ell\mathbb{Z}$, $\ell \leq k$ such that for any $A \in \beta + p^\ell *$, we have $f(A) \equiv 0 \mod p^k$. The *length* of this representative root is the number of precision coordinates of the fixed part β, which is ℓ.

For more properties of representative roots, we direct the reader to [Pan95, BLQ13, DMS21, GCM21].

Solving univariates simultaneously. Using this compact notation, we present the standard Algorithm 3 to find *all* the roots of a univariate polynomial $f(x) \in \mathbb{Z}/p^k\mathbb{Z}$; which is due to [Pan95, BLQ13, DMS21]. However, as required in this paper, we give a slight modification where we solve a system of univariates modulo p^k. The algorithm starts with the input array $(f_1, \ldots, f_r, p, k, \ldots, k)$. This can be seen as a slight modification of the Root-Find algorithm ([DMS21, Algorithm 1]) where instead of looping only over the roots of the polynomial, we loop over the common roots in order to find a root of *all* the polynomials in the system.

Algorithm 3 Root finding of $f_1(x), \ldots, f_r(x) \bmod p^k$

```
1: procedure Root-Find-BLQ($f_1, \ldots, f_r, p, k_1, \ldots, k_r$)
2:     if $r = 0$ then return $*$
3:     if $\exists i$ such that $f_i(x) \equiv 0 \bmod p^{k_i}$ or $k_i = 0$ then
4:         return Root-Find-BLQ($f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_r, p, k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_r$)
5:     $R :=$ roots of $\gcd\{f_i(x) \bmod p \mid i \in [r]\}$ [Eg. use Cantor-Zassenhaus' algorithm [CZ81]].
6:     if $R == \phi$ then return $\phi$
7:     $S := \phi$
8:     for $a \in R$ do
9:         $\tilde{f}_i(x) := p^{v_i} f_i(a + px)\ \forall i \in [r]$, where $v_i = v_p(f_i(a + px))$.
10:        $R_a := $ Root-Find-BLQ($f_1, \ldots, f_r, p, k - v_1, \ldots, k - v_r$)
11:        $S := S \cup (a + pR_a)$
12:    return $S$
```

The correctness of Algorithm 3 directly follows from [BLQ13, Corollary 4], where they prove the correctness for a single polynomial. [BLQ13, Corollary 4] also states that the number of representative roots is at most d many when only a single polynomial is considered.

Theorem A.4. *Algorithm 3 runs in randomized* $\mathsf{poly}(\max_i \deg(f_i), \log p, k)$ *time and returns at most d-many representative roots.*

Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur
Email address: sayaksc@gmail.com

Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur
Email address: nitin@cse.iitk.ac.in