Efficiently Computing Real Roots of Sparse Polynomials

Gorav Jindal
Max-Planck-Institut für Informatik
Campus E1 4
Saarbrücken 66123, Germany
gjindal@mpi-inf.mpg.de

Michael Sagraloff
Max-Planck-Institut für Informatik
Campus E1 4
Saarbrücken 66123, Germany
msagralo@mpi-inf.mpg.de

ABSTRACT

We propose an efficient algorithm to compute the real roots of a sparse polynomial $f \in \mathbb{R}[x]$ having k non-zero real-valued coefficients. It is assumed that arbitrarily good approximations of the non-zero coefficients are given by means of a coefficient oracle. For a given positive integer L, our algorithm returns disjoint disks $\Delta_1, \ldots, \Delta_s \subset \mathbb{C}$, with $s < 2k$, centered at the real axis and of radius less than 2^{-L} together with positive integers $\mu_1, \ldots, \mu_s$ such that each disk Δ_i contains exactly μ_i roots of f counted with multiplicity. In addition, it is ensured that each real root of f is contained in one of the disks. If f has only simple real roots, our algorithm can also be used to isolate all real roots.

The bit complexity of our algorithm is polynomial in k and $\log n$, and near-linear in L and τ, where $2^{-\tau}$ and 2^{τ} constitute lower and upper bounds on the absolute values of the non-zero coefficients of f, and n is the degree of f. For root isolation, the bit complexity is polynomial in k and $\log n$, and near-linear in τ and $\log \sigma^{-1}$, where σ denotes the separation of the real roots.

1 INTRODUCTION

In this paper, we study the problem of computing the real roots of a sparse polynomial

$$f(x) = \sum\nolimits_{i=1}^{k} f_i x^{e_i} \in \mathbb{R}[x], \tag{1.1}$$

where e_i are non-negative integers, with $0 \leq e_1 < e_2 < \ldots < e_k \leq n$, and $2^{-\tau} \leq |f_i| \leq 2^{\tau}$ for all i. We call such a polynomial f an (n, k, τ)*-nomial* or simply a $k-$*nomial* if n and τ are either not specified or clear from the context. We may assume that $k \geq 2$ and $e_1 = 0$ as $1-$nomials do not have any real root different from 0 and as $f \cdot x^{-e_1}$ has exactly the same roots as f except for a possible root at 0. We further assume that, as input, we receive the exponents e_i as well as approximations $\tilde{f}_i$ of the non-zero coefficients f_i. More specifically, we assume the existence of a coefficient oracle that, for any positive integer κ, provides dyadic approximations $\tilde{f}_i = \frac{m_i}{2^{\kappa+1}}$, with $m_i \in \mathbb{Z}$ and $|f_i - \tilde{f}_i| < 2^{-\kappa}$ for all i. We call such an approximation $\tilde{f} = \sum_{i=1}^{k} \tilde{f}_i x^{e_i}$ an *(absolute)* $\kappa-$*bit approximations of* f. Notice that the numbers n and k are directly part of the input, whereas this is not the case for τ. However, we may easily compute (i.e. for a cost bounded by $\tilde{O}(k\tau)$) a good approximation $\tilde{\tau} \in \mathbb{Z}$ of τ with $\tau < \tilde{\tau} < \tau + 2$ by asking the oracle for an κ-bit approximations $\tilde{f}$ of f for $\kappa = 1, 2, 4, \ldots$ until $|\tilde{f}_i| > 2^{-\kappa+1}$ for all i. Then, $\tilde{\tau} := \max_i \lceil |\log |\tilde{f}_i|| \rceil$ fulfills the above inequality.

Within recent years, the problem of *isolating* all (real) roots of a (square-free) polynomial has attracted a lot of interest in the literature; e.g. consider [3, 11, 18] and the references therein. The most efficient algorithms [3, 9, 12, 13, 18] for root isolation achieve running times that are considered to be near-optimal for dense polynomials (i.e. if k is of comparable size as n) $f \in \mathbb{R}[x]$. For polynomials with integer coefficients, the best known bound on the bit complexity of this problem is of size $\tilde{O}(n^2\tau)$. The additional cost for refining isolating intervals to a size less than $2^{-\tau}$, and thus for computing L-bit approximations of all real roots, is $\tilde{O}(n\tau)$; e.g. see [8, 12, 15, 18]. Notice that, for $k-$nomials with integer coefficients, the above bounds are not polynomial in the size of the sparse input representation of $\tilde{f}$, which is bounded by $O(k(\log n + \tau))$ as we need $\log n$ bits to store each exponent e_i and $\tau + 1$ bits to store each f_i. Hence, it is natural to ask whether there exists an algorithm for either root isolation or approximation that runs in polynomial time in the size of the sparse input representation. In [6], Cucker et al. showed how to compute all integer roots of a sparse integer polynomial in polynomial time. Lenstra [10] further improves upon this result giving a polynomial time algorithm to compute all rational factors of f of a fixed constant degree. Furthermore, for polynomials with only a very few non-zero coefficients, there exist polynomial time algorithms to approximate (and also count) the real roots of f. Rojas and Ye [16, 20] propose an algorithm for 3-nomials that uses only $O(\log n)$ arithmetic operations in the field over $\mathbb{Q}$ generated by the coefficients of f. Bastani et al. [2] propose a polynomial time algorithm to count the number of real roots for most 4$-$nomials.

For isolating the roots of a sparse integer polynomial, we recently proposed a method [17] that has polynomial arithmetic complexity and whose bit complexity is $\tilde{\Omega}(n\tau \cdot k^4)$. The latter bound is also near-optimal for small k as there exists a family of Mignotte-like 4$-$nomials, for which the output complexity is always lower bounded by $O(n\tau)$. This result already rules out the existence of a polynomial time algorithm for isolating the roots of a sparse polynomial, however, it remains an open question whether counting the real roots or computing $L-$bit approximations of the real roots can be achieved in polynomial time.

In this paper, we give a positive answer for a slight relaxation of the latter problem. That is, we give a polynomial time algorithm

ISSAC '17, July 25-28, 2017, Kaiserslautern, Germany

ACM ISBN 978-1-4503-5064-8/17/07…$15.00
http://dx.doi.org/10.1145/3087604.3087652

to compute a partial clustering of the roots that contains all real roots of f. For a more precise statement, we need the following definitions, where $\Delta_r(m) \subset \mathbb{C}$ denotes the open disk in complex space with center m and radius r.

Definition 1 ((L, I)-covering)**.** For a polynomial f as in (1.1), an integer $L \in \mathbb{N}$, and an interval $I \subset \mathbb{R}$, we call a list $((\Delta_{r_1}(m_1), \mu_1), (\Delta_{r_2}(m_2), \mu_2), \ldots, (\Delta_{r_t}(m_t), \mu_t))$ an *(L, I)-covering for f* if the following conditions are fulfilled:

(1) The disks $\Delta_{r_i}(m_i)$ are pairwise disjoint, m_j are real values with $m_1 < \cdots < m_t$, and $r_j \leq 2^{-L}$ for all j.
(2) $\Delta_{r_j}(m_j)$ contains exactly μ_j roots of f for all j.
(3) For every real root ξ of f in I, there exists some disk $\Delta_{r_j}(m_j)$ that contains ξ.

We further introduce a weaker version of L-covering:

Definition 2 (Weak (L, I)-covering)**.** A *weak (L, I)-covering* for f is a list $(I_1, \ldots, I_t)$ of open disjoint and sorted real intervals that fulfills the following conditions:

(1) The width of each interval I_j is at most 2^{-L}.
(2) For every real root ξ of f in I, there exists an interval I_j that contains ξ.

If $I = \mathbb{R}$, we omit I and just call a (weak) *$(L, \mathbb{R})$-covering for f* a (weak) *L-covering for f.* Then, our main contribution is a polynomial-time algorithm for computing an $L-$covering:

Theorem 3. *For an (n, k, τ)-nomial, we can compute an L-covering $\mathcal{L}$ of size $|\mathcal{L}| < 2k$ in time $\tilde{O}(\text{poly}(k, \log n) \cdot (\tau + L))$.*

Notice that our algorithm computes $L-$bit approximations of all real roots but might also return (real-valued) $L-$bit approximations of some non-real roots with a small imaginary part. Further notice that unless μ_j is odd, we also do not know whether m_j actually approximates a real root, and unless $\mu_j = 1$, we cannot conclude that a disk $\Delta_{r_j}(m_j)$ in an L-covering is isolating for a root of f. Hence, in general, our algorithm does not yield the correct number of distinct real roots. However, if f has only simple roots, we may compute an $L-$covering for f for $L = 2, 4, 8, \ldots$ until $\mu_j = 1$ for all j. Then, the disks $\Delta_{r_i}(m_i)$ isolate all real roots.

Theorem 4. *Let f be an (n, k, τ)-nomial with only simple real roots, and let σ be the minimal distance between any two (complex) distinct roots of f (i.e. the* separation *of f). Then, we can compute isolating intervals for all real roots in $\tilde{O}(\text{poly}(k, \log n)(\tau + \log \max(1, 1/\sigma)))$ bit operations.*

We improve upon [17] in several ways. Namely, [17] only applies to integer polynomials, whereas our novel approach applies to polynomials with arbitrary real coefficients. In addition, the running time of the algorithm in [17] does not adapt to the actual hardness of the roots, whereas the complexity of our novel approach rather depends on the actual separation than on the worst-case bound [19] of size $2^{-O(n(\tau + \log n))}$ for the separation of an integer polynomial. In the worst case, our method isolates all real roots of a very sparse integer polynomial (i.e. $k = (\log(n\tau))^{O(1)}$) in time $\tilde{O}(n\tau)$, and is thus near optimal.; see [17]

Remark. Due to space limitations, we omitted proofs of some results, which can be found in the full version [7] of the paper on arXiv.

Overview of the Algorithm. Before we go into detail, we give a brief overview of our algorithm, where we omit technical details. We first remark that the problem of computing an $(L, [1, \infty))$-covering can be reduced to the problem of computing an $(L, [0, 1])$-covering (in fact, we are computing an $(L, [0, 1 + 1/n])$-covering but this for technical reasons only) by means of the coordinate transformation $x \mapsto \frac{1}{x}$ followed by multiplication with x^n. We may also reduce the problem of computing an $(L, (-\infty, 0])$-covering of f to the problem of computing an $(L, [0, \infty))$-covering by means of the coordinate transformation $x \mapsto -x$. Hence, we are eventually left with merging $(L, [0, 1])$-coverings for the polynomials f, $x^n \cdot f(1/x)$, $f(-x)$, and $x^n \cdot f(-1/x)$ in a suitable manner. We give details for this step in Section 7. Notice that the considered coordinate transformation preserves the sparseness of the input polynomial, hence we may concentrate on the problem of computing an $(L, [0, 1])$-covering for f only. For this, we first compute a weak $(L, [0, 1])$-covering of f, which is achieved by recursively computing weak $(L, [0, 1])$-coverings of the so-called *fractional derivatives of f*.

Definition 5 (Fractional Derivatives)**.** Let f be a polynomial as in (1.1). Then, we define $f^{[1]} := \frac{f'}{x^{e_2 - 1}}$ as the *(first) fractional derivative of f*. In other words, we divide the first derivative f' of f by the highest possible power of x that divides f'. The *$i-th$ fractional derivative $f^{[i]}$ of f* is then recursively defined as the first fractional derivative of $f^{[i-1]}$. Notice that, for $i \leq k - 1$, $f^{[i]}$ is an $(n, k - i, \tau + k \cdot \log n)-$nomial with a non-zero constant term and $f^{[i]} \equiv 0$ for $i \geq k$. We further use the notation $\mathcal{D}_f$ to denote the tuple of all non-zero fractional derivatives $f, f^{[1]}, f^{[2]}, f^{[2]}, \ldots, f^{[k-1]}$, i.e, $\mathcal{D}_f = (f, f^{[1]}, f^{[2]}, f^{[3]}, \ldots, f^{[k-1]})$.

The general idea of recursively computing the real roots of f from the real roots of its fractional derivatives has already been considered in previous work; e.g. [1, 4, 5, 10, 14, 16, 17]. The simple idea is that, given a weak $(L, [0, 1])$-covering $(I_1', \ldots, I_{t'}')$ for $f^{[1]}$, we already know that in between two consecutive intervals $I_j = (a, b)$ and $I_{j+1} = (c, d)$, the polynomial f is monotone, and thus there can be at most one real root in between b and c, which then must be simple. In order to check for the existence of such a root, it suffices to check whether f changes signs at the points b and c. In case of a sign change, we may then refine the interval (b, c), which is known to be isolating for a real root of f, to a width less than 2^{-L}. If we proceed in this way for all intervals in between two consecutive intervals as well as with the leftmost interval, whose endpoints[1] are 0 and the left endpoint of I_1', and the rightmost interval, whose endpoints are the right endpoint of $I_{t'}'$ and 1, then we obtain a set of intervals I_j'' of size at most 2^{-L} that cover all real roots of f that are contained in $[0, 1]$ but in none of the intervals I_j'. Hence, the union of the intervals I_j' and I_j'' constitutes an $(L, [0, 1])$-covering for f. This shows how to compute an $(L, [0, 1])$-covering for f from recursively computing $(L, [0, 1])$-coverings for its fractional derivatives.

[1] For technical reasons, we will indeed consider slight perturbations of 0 and 1 in our algorithm.

We remark that, in this simplistic description, we have omitted several key problems one faces when formalizing the algorithm: Evaluating the sign of a polynomial f at given points b, c may require a very high precision, which should be avoided to ensure a polynomial bit complexity. In addition, we need an efficient refinement method that uses only a polynomial number of iterations. For the latter problem, we use a slightly modified variant of our algorithm from [17, 18]. For the computation of the sign of f (and its higher order fractional derivatives) at certain points, we consider an approach that allows us to slightly perturb the evaluation points such that the absolute value of each of the considered polynomials does not become too small. One major contribution of this paper, when compared to our previous work [17], is to show that this can be done in way such that the precision always stays polynomial in $\log n$, k, τ, and L.

In the second step, we derive an $(L, [0, 1])$−covering from a weak $(L', [0, 1])$−covering, where L' has been chosen sufficiently large. A straight forward approach would be to use a method for computing the number of roots in the *one-circle region* $\Delta(I) = \Delta_r(m)$ of each interval I in the weak $(L', [0, 1])$−covering. Here, $\Delta(I)$ is defined as the disk centered at the midpoint $m = m(I)$ of I and passing through the endpoints of the interval. In the literature, several methods have been proposed to count the number of roots in a disk in complex space. Unfortunately, these algorithm are not sparsity aware, which rules out a straight-forward application of them. Recent work [3] introduces the so-called T_l-test, a method for root counting based on Pellet’s Theorem. The method only needs to compute approximations of the coefficients of the polynomial $f(m + r \cdot x)$, however, we cannot afford to compute all coefficients. Fortunately, in our situation, only the first k^2 coefficients are actually needed to determine the outcome of the test. In order to guarantee success of the test, it may further be necessary to merge some of the intervals in the weak covering and to consider disks that are larger than the one-circle regions of the merged intervals. This explains why we need a weak $(L', [0, 1])$−covering with a sufficiently large $L' > L$. We consider our method for counting the roots of a sparse polynomial in a disk as the second main contribution of our paper.

2 ON THE GEOMETRY OF ROOTS

Descartes’ Rule of Signs states that the number $\text{var}(F)$ of sign changes in the coefficient sequence of a polynomial $F \in \mathbb{R}[x]$ constitutes an upper bound on the number of positive real roots. Hence, it follows immediately that a k−nomial f as in (1.1) has at most $k - 1$ negative and at most $k - 1$ positive real roots. Apart from this simple fact, k-nomials have indeed much more structure on their roots, which we will briefly survey in this section.

Let $I = (a, b)$ be an interval, $F_I(x) := (x + 1)^n \cdot F\left(\frac{ax+b}{x+1}\right)$, and $v_I := \text{var}(F, I)$ be the number of sign changes in the coefficient sequence of the polynomial F_I. Notice that there is a one-to-one correspondence between the roots of F in I and the positive real roots of F_I via the Möbius transformation that maps a point $x \in \mathbb{C} \setminus \{-1\}$ to $\frac{ax+b}{x+1} \in \mathbb{C}$. Thus, v_I constitutes an upper bound on the number of roots of F in I. In fact, v_I also constitutes a lower bound on the number of roots in the so called *Obreshkoff lens* L_n of the interval I. L_n is defined as the intersection $L_n := \overline{C}_n \cap \underline{C}_n$ of the two open disks $\overline{C}_n, \underline{C}_n \subset \mathbb{C}$ that intersect the real axis in the endpoints a and b of I, and whose centers see the line segment (a, b) under the angle $\frac{2\pi}{n+2}$. For an illustration, see [18, Fig. 1]. It further holds [17, 18]) that $\text{var}(F, I) \leq \text{var}(F) \leq k - 1$ for any interval $I \subset \mathbb{R}^+$, hence we conclude that the Obreshkoff lens L_n of any such interval contains at most $k - 1$ roots. For $b \mapsto \infty$, the Obreshkoff lens L_n of the interval $I = (0, b)$ converges to the cone C_n whose boundary are the two half-lines starting at the origin and intersecting the real axis at an angle $\pm\frac{\pi}{n+2}$; see Figure 2.1. Hence, it follows that the interior of C_n contains at most $k - 1$ roots of any given k−nomial of degree n.

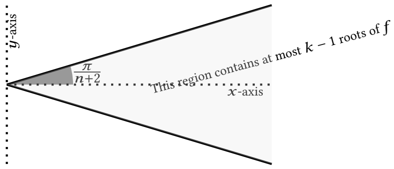

Figure 2.1: The cone C_n contains at most $k - 1$ roots of f.

Theorem 6. *The cone C_n contains at most $k - 1$ roots of any k-sparse polynomial of degree n.*

3 POLYNOMIAL ARITHMETIC

Our algorithm only needs to perform basic operations on polynomials. In particular, we need to evaluate the sign of a given sparse polynomial at some points x. As we already mentioned in the overview of our algorithm, the complexity of this operation becomes too large if the value of the polynomial at a given point x is almost zero as then one needs to perform computations with a very high precision. Also, exact evaluation of a sparse polynomial at a rational point (even of small bitsize) is expensive as the output has bitsize linear in n. Instead, we consider approximate evaluation, which allows us to evaluate a sparse polynomial f as in (1.1) at an arbitrary point $x \in (0, 1 + 1/n)$ to an absolute error less than 2^{-L} in a time that is polynomial[2] in $\log n$, k, τ, and L. More precisely, we derive the following result:

Lemma 1. *Let $f \in \mathbb{R}[x]$ be an (n, k, τ)-nomial, c be a positive real number, and L a non-negative integer. Then, we can compute an L-bit approximation λ of $f(c)$ (i.e. $|\lambda - f(c)| < 2^{-L}$) in a number of bit operations bounded by*

$$\tilde{O}((k + \log n) \cdot (L + n \log \max(1, |c|) + \log n + \tau + k)).$$

We already mentioned that evaluating the sign of a polynomial f at a point x might be costly if $f(x)$ has a small absolute value. In order to avoid such undesired computations, we first perturb x in a suitable manner. That is, instead of evaluating the sign of f at x, we evaluate its sign at a nearby point, where f becomes large enough. This can be done in a way such that the actual behavior of the algorithm does not change. We will call such points “admissible”. We remark that we already used this concept in previous work [17, 18]. Here, we modify the approach to choose an admissible point, where the sign of each fractional derivative of a sparse polynomial f can be evaluated in polynomial time.

[2] Notice that, for $c \in (0, 1 + 1/n^{O(1)})$, we may omit the term $n \log \max(1, |c|)$ in the bounds stated in Lemma 1.

Definition 7 (Admissible point). Let $g : \mathbb{R} \to \mathbb{R}$ be a function and $m[t;\delta] = \{m_i := m + (i-t)\cdot\delta; i = 0, 1, \ldots, 2t\}$ be a multipoint. Then, we call a point $m^* \in m[t;\delta]$ to be $(g, m[t;\delta])$-*admissible* if $|g(m^*)| \geq \frac{1}{8}\cdot \max_{x\in m[t;\delta]} |g(x)|$.

If t and δ (or even m and g) are clear from the context, we simply call a $(g, m[t;\delta])$-admissible point (g, m)-admissible (or just admissible). Since the value of g at an admissible points is "relatively large", we expect that g has no root in a corresponding neighborhood.

Lemma 2. *Let $m^* \in m[t;\delta]$ be an $(f, m[t;\delta])$-admissible point for an (n,k,τ)-nomial f, with $m \in \mathbb{R}_+$ and $2 \leq k \leq t \leq k^2$. If $\frac{m}{\delta} > 4k^2n^2$, then the disk $\Delta_{\delta\cdot k^{-4k}}(m^*)$ contains no root of f.*

Definition 8. Let $\mathcal{G} = (g_1, g_2, \ldots, g_t)$ be a tuple of t functions $g_i : \mathbb{R} \to \mathbb{R}$. Then, $M_{\mathcal{G}}(x)$ is defined as follows:

$$M_{\mathcal{G}}(x) := \min(|g_1(x)|, |g_2(x)|, , \ldots, |g_t(x)|).$$

For a fixed real x, we call $\tilde{\mathcal{G}}(x) = (\tilde{g}_1(x), \tilde{g}_2(x), \ldots \tilde{g}_t(x))$ an *L-approximation of $\mathcal{G}(x)$* if $|\tilde{g}_i(x) - g_i(x)| \leq 2^{-L}$ for all i.

We first show how to compute an admissible point $m^* \in m[t;\delta]$ for $M_{\mathcal{G}}(x)$ under the assumption that we can compute an L-approximation of $\mathcal{G}(x)$ for any $x \in m[t;\delta]$ in time $T(L)$.

Lemma 3. *Let $\mathcal{G} = (g_1, g_2, \ldots, g_t)$ be as in Definition 8, $m[t;\delta]$ a multipoint and $\lambda := \max_{a\in m[t;\delta]} \left|M_{\mathcal{G}}(a)\right|$. Suppose that for a point $m_i \in m[t;\delta]$ we can compute an L-approximation of $\mathcal{G}(m_i)$ in time $T(L)$. Then we can compute an $(M_{\mathcal{G}}, m[t;\delta])$-admissible point $m^* \in m[t;\delta]$ as well as an integer ℓ^* with $2^{\ell^*-1} \leq \left|M_{\mathcal{G}}(m^*)\right| \leq \lambda \leq 2^{\ell^*+1}$. in time $O(t\cdot \log\log\max(\lambda^{-1}, 1)\cdot (T(\log\max(\lambda^{-1}, 1)))$.*

We now apply the above lemma to $\mathcal{G} := \mathcal{D}_f$, the sequence of fractional derivatives of f. Then, Lemma 1 yields a bound of the bit complexity of computing L-approximations of $\mathcal{D}_f(m_i)$ for all $m_i \in m[t;\delta]$, which directly depends on $\lambda := \max_{m_i\in m[t;\delta]} \left|M_{\mathcal{D}_f}(m_i)\right|$.

Corollary 9. *Assume that $f(x)$ is a (n,k,τ)-nomial, $m[t;\delta]$ a multipoint and $\lambda := \max_{m_i\in m[t;\delta]} \left|M_{\mathcal{D}_f}(m_i)\right|$. Further assume that $m[t;\delta] \subset (0, \alpha)$ for some positive real α. Then, we can determine an $(M_{\mathcal{D}_f}, m[t;\delta])$-admissible point m^* and an integer ℓ^* with*

$$2^{\ell*-1} \leq \left|M_{\mathcal{D}_f}(m^*)\right| \leq \lambda \leq 2^{\ell^*+1}$$

using $\tilde{O}(t\cdot k\cdot(k+\log n)\cdot(\tau+k\log n+n\log\max(1,\alpha)+\log\max(1,\lambda^{-1})))$ many bit operations.

The following bound on λ implies that, for suitably chosen t,m and δ, we can compute m^* in polynomial time.

Lemma 4. *Let $f \in \mathbb{R}[x]$ be a (n,k,τ)-nomial as in (1.1), and let a, r be positive real numbers with $r < a$ and such that $(a-r, a+r)$ does not contain any real root of any fractional derivative of $f(x)$. Then,*

$$|M_{\mathcal{D}_f}(a)| = 2^{-O(k(k\log n+\tau+\log\max(1,\frac{1}{r})+n\log\max(1,a+r)))}.$$

Proof. We may assume that r is small enough to guarantee that $\frac{a}{r} > 2n$. This implies that, for any two points $x, x' \in I_1 := (a-r, a+r)$, we have that $x/x' \in (1-1/n, 1+1/n)$. Now, let us write $f = c + x^j\cdot g$ with a constant c of absolute value at least $2^{-\tau}$ and g an $(n-j, k-1, \tau+\log n)$–nomial that is not divisible by x. Then, it holds that $f^{[1]} = j\cdot g + x\cdot g'$, and thus $f' = x^{j-1}\cdot f^{[1]}$. In addition, since $I_1 := (a-r, a+r)$ does not contain any root of f and $f^{[1]}$, it follows that f is monotone on I and only takes positive or negative values. This implies that $|f(t) - f(t')| = ||f(t)| - |f(t')||$ for all $t, t' \in I$. In addition, for any $t \in I_2 := (a - r/2, a + r/2)$, we can choose a point $t' = t \pm r/2$ such that $|f(t)| > |f(t')|$. Now, according to the mean value theorem, there exists a ξ in between t and t' with $f(t) - f(t') = (t-t')\cdot f'(\xi) = \frac{r}{2}\cdot\xi^{j-1}\cdot f^{[1]}(\xi)$. Hence, we obtain $|f(t)| > |f(t)| - |f(t')| = ||f(t)| - |f(t')|| = |f(t) - f(t')| \geq \frac{r}{2}\cdot\xi^{j-1}\cdot f^{[1]}(\xi) \geq \frac{r}{8}\cdot t^{j-1}\cdot f^{[1]}(\xi)$, where the latter inequality follows from $(\xi/t)^{j-1} > (1-1/n)^n > 1/2$. Also, $|f(t)| \geq |c| - t^j\cdot|g(t)| \geq 2^{-\tau} - t^{j-1}\cdot k\cdot 2^{\tau+\log n}\cdot\max(1,a+r)^n$. With $\varepsilon := \min(1, \inf_{x\in I_1}|f^{[1]}(x)|)$, the above inequalities thus imply that

$$|f(t)| > \max(\frac{r\varepsilon}{8}\cdot t^{j-1}, 2^{-\tau} - t^{j-1}k2^{\tau+\log n}\cdot\max(1,a+r)^n)$$

If $t^{j-1} < 2^{-\tau-1}(k2^{\tau+\log n}\cdot\max(1,a+r)^n)^{-1}$, the second argument in the above term becomes larger than $2^{-\tau-1}$. Otherwise, the first term becomes larger than $\frac{r\varepsilon}{8}\cdot 2^{-\tau-1}(k2^{\tau+\log n}\cdot\max(1,a+r)^n)^{-1}$. Thus, we have $\inf_{x\in I_2}|f(x)| > r\cdot\varepsilon\cdot 2^{-2\tau-1-2\log n-n\log\max(1,a+r)}$. We now recursively apply the above result to the fractional derivatives $f^{[k-i]}$ and the intervals $I_i := (a - \frac{r}{2^{i-1}}, a + \frac{r}{2^{i-1}})$, where $i = 1, 2, \ldots, k$. Notice that each of the polynomials is an $(n, k, \tau + k\log n)$–nomial and that $\inf_{x\in I_1}|f^{[k-1]}(x)| > 2^{-\tau}$ as $f^{[k-1]}$ is a constant of absolute value at least $2^{-\tau}$. Hence, it follows that

$$\inf_{x\in I_i}|f^{[k-i]}(x)| > 2^{-\tau-i\cdot(2\tau-1-2k\log n-n\log\max(1,a+r))}\cdot\prod_{j=1}^{i-1}\frac{r}{2^j}.$$

□

Combining the above lemma and Corollary 9 now yields

Theorem 10. *Let f be a (n,k,τ)-nomial as in (1.1), and let $m[t;\delta]$ be a multipoint with $t \geq k^2$ and $m[t;\delta] \subset (0,\alpha)$ for some for some real number α. Then, we can compute an $(M_{\mathcal{D}_f}, m[t;\delta])$-admissible point m^* using $\tilde{O}(t\cdot k^2\cdot(k+\log n)\cdot(k\log n+\tau+\log\max(1,\frac{1}{\delta})+n\log\max(1,\alpha)))$ bit operations.*

Proof. Since each fractional derivative of f has at most $k-1$ positive real roots and since $t \geq k^2$, there exists an $a \in m[t;\delta]$ such that $(a-\delta/2, a+\delta/2)$ does not contain any real root of any of fractional derivative. Hence, Lemma 4 implies that $\lambda := \max_{x\in m[t;\delta]}|M_{\mathcal{D}_f}(x)| \geq |M_{\mathcal{D}_f}(a)|$ is lower bounded by $2^{-O(k(k\log n+\tau+\log\frac{1}{\delta}+n\log\max(1,a+\delta)))}$. Corollary 9 then yields the claimed bound on the running time. □

4 REFINEMENT

A crucial subroutine of our overall algorithm is an efficient method for refining an interval $I_0 = (a_0, b_0) \subset \mathbb{R}_+$, with $\max(|\log a_0|, |\log b_0|) = O(\tau)$, that is known to be isolating for a simple real root of a k-nomial f. It is assumed that the algorithm receives the sign of f at the endpoints of I_0 as additional input. For the refinement, we consider the algorithm NewRefine from Section 3 in [17] (see also Section 5 in [18]), however, we make a single (minor) modification. As the argument from [17] directly applies, we only state the main results and refer the reader to [17] for details.

NewRefine recursively refines I_0 to a size less than 2^{-L} using a trial and error approach that combines Newton iteration and bisection. For this, only f and its first derivative f' need to be evaluated. More precisely, in each iteration, the algorithm computes $(f, m[\lceil k/2\rceil;\delta])$–admissible points m^* for a constant number of points $m \in I$ and a corresponding δ of size $2^{-O(\tau+\log n+L)}$. In addition, f and f' are evaluated at these admissible points to an absolute precision that is bounded by $O(\log\max(1,|f(m^*)|^{-1})+\log n+L+\tau)$. Each endpoint of the interval returned by NewRefine is then either one of the admissible points computed in a previous iteration or one of the endpoints of I_0.

We now propose the following modification of NewRefine, which we denote NewRefine*: Whenever NewRefine asks for an $(f, m[\lceil k/2\rceil;\delta])$–admissible point m^*, we compute an $(M_{\mathcal{D}_f}, m[k^2;\delta'])$–admissible point m^*, with $\delta' = \delta\cdot\frac{\lceil k/2\rceil}{k^2}$, instead. Then, the same argument[3] as in [17] yields:

Theorem 11. *For refining I_0 to a size less than 2^{-L}, the algorithm NewRefine* needs $O(k\cdot(\log n+\log(\tau+L)))$ iterations. In each iteration, we need to compute a constant number of $(M_{\mathcal{D}_f}, m[k^2;\delta'])$–admissible points m^*, with $m[k^2;\delta']\subset I_0$ and $\delta' = 2^{-O(\tau+\log n+L)}$. In addition, the polynomials f and f' have to evaluated at m^* to an absolute precision bounded by $O(\log\max(1,|f(m^*)|^{-1})+\log n+L+\tau)$.*

Combining Theorems 11 and 10, we obtain a bound on the complexity of refining I_0 to a size less than 2^{-L}:

Corollary 12. *For refining I_0 to a size less than 2^{-L}, the algorithm NewRefine* needs*

$$\tilde{O}(k^5\cdot(k+\log n)\cdot\log n\cdot(k\log n+\tau+L+n\log\max(1,b_0)))$$

bit operations. For each endpoint p of the interval returned by NewRefine, it holds that

$$M_{\mathcal{D}_f}(p) = 2^{-O(\ell+k(k\log n+\tau+L+n\log\max(1,b_0)))}.$$

with $\ell := \log\min(1, M_{\mathcal{D}_f}(a_0), M_{\mathcal{D}_f}(b_0))^{-1}$.

5 COMPUTING A WEAK COVERING

We now describe how to compute a weak $(L,[0,1+1/n])$-covering for a given (n,k,τ)-nomial f in polynomial time. We first compute an upper bound $\tilde{\tau}\in\mathbb{Z}$ for τ with $\tau\le\tilde{\tau}\le\tau+2$, and define $\delta := \min(2^{-2\tilde{\tau}-2}, 1/n)\cdot k^{-2}$. Then, in the first step, we compute $(M_{\mathcal{D}_f}, m[k^2;\delta])$–admissible points a^* and b^* for $m := 2^{-2\tau-2}$ and $m := 1+2/n$, respectively. Then, we follow the approach as outlined on page 2 to compute a weak $(L,[a^*,b^*])$-covering for f, where we use the algorithm NewRefine* from the previous Section to refine isolating intervals for the roots of the fractional derivatives of f to a size less than 2^{-L}. The so obtained covering is indeed also a weak $(L,[0,1+1/n])$-covering for f, which follows from the fact that $b^*\ge 1+1/n$ and each positive root of f is lower bounded by $(1+\max_{i=1}^k|f_i|/|f_1|)^{-1}$ due to Cauchy's root bound [19]. For details, consider the exact definition of Algorithm 1.

Algorithm 1 Compute a weak $(L,[0,1])$-covering of f

```
Input  : An $(n,k,\tau)$-nomial $f$ and an $L\in\mathbb{N}_0^+$.
Output : A weak $(L,[0,1+1/n])$-covering of $f$.
Compute $\tilde{\tau}\in\mathbb{N}$ with $\tau\le\tilde{\tau}\le\tau+2$.
Define $\delta := \frac{1}{k^2}\cdot\min(\frac{1}{n}, 2^{-2\tilde{\tau}-2})$
Compute $(M_{\mathcal{D}_f}, m[k^2;\delta])$–admissible points $a^*$ and $b^*$ for
    $m := 2^{-2\tilde{\tau}-2}$ and $m := 1+\frac{2}{n}$, respectively.
Compute the sign of $f$ at $x=a^*$ and $x=b^*$.
for $i = k-1$ to $0$ do
    if $i = k-1$ then
        Compute a trivial weak $(L,[a^*,b^*])$-covering $W_{k-1}$ for
            $f^{[k-1]}$ ($f^{[k-1]}$ has only one monomial).
        $W_{k-1} = \{(a^*,a^*),(b^*,b^*)\}$.
    else
        $W_{i+1}$ = weak $(L,[a^*,b^*])$-covering for $f^{[i+1]}$ computed in
            the previous iteration of this loop.
        $W_i = W_{i+1}$.
        for each consecutive intervals $(a,b)$ and $(c,d)$ in $W_{i+1}$ do
            Compute signs of $f^{[i]}(b)$ and $f^{[i]}(c)$
            if $f^{(i)}(b)f^{(i)}(c) < 0$ then
                Use NewRefine* to refine the isolating interval
                    $(b,c)$ to a new interval $(b',c')$ of length at most
                    $2^{-L}$.
                Compute signs of $f^{[i]}(b')$ and $f^{[i]}(c')$.
                $W_i = W_i\cup(b',c')$
return $W_0$.
```

Correctness of the algorithm follows directly from our considerations on page 2. Further notice that, for each i in the outermost for-loop of the algorithm, we add at most $k-i-1$ intervals to W_i to obtain W_{i+1} as $f^{[i]}$ has at most $k-i-1$ positive real roots. Hence, each list W_i contains at most k^2 many intervals. It remains to bound the running time of Algorithm 1. The proof of the following Lemma follows in a straight forward manner from Theorem 10, Corollary 12, and the fact that we need to call the refinement algorithm at most k times for each fractional derivative.

Lemma 5. *Algorithm 1 computes a weak $(L,[0,1+\frac{1}{n}])$-covering for f consisting of at most k^2 many intervals. Its bit complexity is $\tilde{O}(k^7(k+\log n\cdot(k\log n+\tau+L)\cdot)\log n)$.*

In order to further process a weak $(L,[0,1+1/n])$-covering for f, we need the intervals in the weak covering to be well separated. For given $L,\lambda\in\mathbb{N}_0$, we say that a list $\mathcal{L}$ of intervals is *(L,λ)-separated* if the distance $\operatorname{dist}(I,J)$ between I and its neighboring intervals is at least $\min(2^{-L},\lambda\cdot w(I))$. Notice that, starting from an arbitrary list $\mathcal{L}$ of intervals, we can always deduce an (L,λ)-separated list $\mathcal{L}'$ from $\mathcal{L}$ in a way such that each interval in $\mathcal{L}$ is contained in

[3] The argument in [17] only uses that, in each iteration, we choose an arbitrary point $m^*\in[m-\lceil k/2\rceil\cdot\delta, m+\lceil k/2\rceil\cdot\delta]$.

an interval from $\mathcal{L}'$. Namely, this can be achieved by recursively merging pairs of intervals $I, J \in \mathcal{L}$ that violate the above condition until the actual list is (L,λ)*-separated.* It is easy to see that

$$w(\mathcal{L}') \leq (2+\lambda)^{|\mathcal{L}|} \cdot \max(2^{-L}, w(\mathcal{L})),$$

where $w(\mathcal{L})$ and $w(\mathcal{L}')$ denote the maximal width of an interval in $\mathcal{L}$ and $\mathcal{L}'$, respectively. Hence, by first computing a weak $(L', [0, 1+1/n])$-covering $\mathcal{L}$, with $L' = L + k^2 \cdot \log(2+\lambda)$ and $|\mathcal{L}| = O(k^2)$, and then recursively merging the intervals, we obtain a weak $(L, [0, 1+1/n])$−covering for f that is also (L,λ)-separating and whose intervals have width at most 2^{-L}. From Lemma 5, we thus conclude:

Corollary 13. *For any $\lambda, L \in \mathbb{N}_0$, we can compute a (L,λ)- separating weak $(L, [0, 1+1/n])$−covering for f in $\tilde{O}(k^7(k+\log n) \cdot (k\log n + \tau + L + k^2\log(2+\lambda)) \cdot \log n)$ bit operations.*

6 T_L-TEST

In the previous section, we have shown how to compute a weak $(L, [0, 1+1/n))$-covering of a given (n,k,τ)-nomial f. Now, we aim to convert this weak covering to a covering of f. For this, we need an algorithm to count the number of roots of $f(x)$ contained in a given disk. Recent work [3] introduces a simple corresponding algorithm, denoted T_l-test, which is based on Pellet's Theorem. More precisely, for an arbitrary polynomial $F \in \mathbb{C}[x]$, a disk $\Delta = \Delta_r(m) \subset \mathbb{C}$, and a parameter $K \geq 1$, we consider the inequality

$$T_l(\Delta, K, F): \left|\frac{F^{(l)}(m)r^l}{l!}\right| - K \cdot \sum_{i \neq l} \left|\frac{F^{(i)}(m)r^i}{i!}\right| > 0. \qquad (6.1)$$

Hence, we check whether the absolute value of the l-th coefficient a_l of $F_\Delta(x) = f(m+rx) = \sum_{i=0}^{n} a_i x^i$ dominates the sum of the absolute values of all remaining coefficients weighted by the parameter K. We say that $T_l(\Delta, K, F)$ succeeds if the above inequality is fulfilled. Otherwise, we say that it fails. In case of success (for any $K \geq 1$), Δ contains exactly l roots of F counted with multiplicity, whereas we have no information in case of a failure. However, in [3], we derive sufficient conditions on the success of the T_l-test:

Theorem 14 ([3], Corollary 1)**.** *Let $F \in \mathbb{C}[x]$ be a polynomial of degree n, and $\Delta_r(m)$ be a disk. If $\Delta_r(m)$ as well as the enlarged disk $\Delta_{256n^5r}(m)$ contain l roots of F counted with multiplicity, then* $T_l(\Delta_{16nr}(m), \frac{3}{2}, F)$ succeeds.

Unfortunately, the above test has two major drawbacks when dealing with sparse polynomials. First, we need to compute the coefficients F_Δ exactly, which we cannot afford as the bitsize of each coefficient is at least linear in n. Second, an even more severe, there are n coefficients to be computed. Hence, using the above approach directly to count the number of roots of a sparse polynomial f does not work. Instead, we propose two modifications to overcome these issues. The first modification, namely to use approximate (in a proper manner) instead of exact arithmetic, has already been considered in previous work. However, the second modification is more subtle. It exploits the fact that, for a suitably chosen disk centered at some admissible point, only the first k^2 coefficient are relevant for the outcome of the above test.

We first go into details with respect to our first modification. Let us define $E_\ell := |a_l|$ and $E_r := K \cdot \sum_{i\neq l} |a_i|$ the expressions on the left and right hand side of the inequality in (6.1). We aim to check whether $E_\ell - E_r > 0$ or not. In general, if a predicate $\mathcal{P}$ is of the latter form $\mathcal{P} = (E_\ell - E_r > 0)$ with two (computable) expressions E_ℓ and E_r, you can compute approximations $\tilde{E}_\ell$ and $\tilde{E}_r$ of E_ℓ and E_r with $|\tilde{E}_\ell - E_\ell| < 2^{-L}$ and $|\tilde{E}_r - E_r| < 2^{-L}$ for $L = 1, 2, 4, \ldots$ For a certain L, you may then try to compare E_ℓ and E_r taking into account their corresponding approximations and the approximation error. Eventually (i.e. for a sufficiently large L), you either succeed, in which case you can return the sign, or assert that E_ℓ and E_r are good approximations of each other. In the latter case, you just return a flag called Undecided. In short, this is the idea of so-called *soft-predicates*. For details, we refer to [3].

```
Algorithm 2 Soft Predicate $\tilde{\mathcal{P}}$
Input  : A predicate $\mathcal{P}$ defined by non-negative expressions $E_\ell$
         and $E_r$ , with $E_\ell \neq 0$ or $E_r \neq 0$; i.e. $\mathcal{P}$ succeeds if and only
         if $E_\ell > E_r$. A rational constant $\delta > 0$.
Output : True, False, or Undecided. In case of True (False), $\mathcal{P}$
         succeeds (fails). In case of Undecided, we have
         $\frac{1}{1+\delta} \cdot E_\ell < E_r \leq (1+\delta) \cdot E_\ell$
```

Notice that, in cases where E_ℓ considerably differs from E_r, the soft predicate $\tilde{\mathcal{P}}$ allows us to compute the sign of $\mathcal{P}$ without the need of exact arithmetic. In all other cases (i.e. if it returns Undecided), we know at least that E_ℓ and E_r are good approximations of each other. We remark that, in [3], the above soft predicate $\tilde{\mathcal{P}}$ was only described for $\delta = \frac{1}{2}$, however, it easily generalizes to any constant δ. In [3, Lem. 2], it has been shown that, for any constant δ, Algorithm 2 needs an L_0-bit approximation of E_ℓ and E_r with L_0 bounded by

$$L_0 \leq 2 \cdot (\max(1, \log\max(E_\ell, E_r)^{-1}) + 4).$$

In [3], we considered a soft-variant of the T_l-test, where we compared the expressions $E_\ell := |a_l|$ and $E_r := \sum_{i\neq l} |a_i|$. Now, we apply the above soft-predicate to the expressions $E_\ell := a_l$ and $E_r := \sum_{i\neq l}^{i\leq k^2} |a_i|$, that is, we replace the entire sum $\sum_{i\neq l} |a_i|$ by its truncation after the first k^2 terms. However, we will make the assumption that the truncated sum is upper bounded by $\frac{|a_0|}{128}$; see Algorithm 3. This might look haphazardly at first sight, however, we will later see that the latter condition is always fulfilled for a k-nomial F and a suitable disk $\Delta_r(m)$ centered at an admissible point.

```
Algorithm 3 $\tilde{T}_l$-test
Input  : An $(n,k,\tau)$-nomial $f(x)$, a disk $\Delta := \Delta_r(m)$ in the
         complex space and an integer $l$ with $0 \leq l \leq k$. It is
         required that $\sum_{i>k^2} |a_i| \leq \frac{|a_0|}{128}$, where
         $f_\Delta(x) = \sum_{i=0}^{n} a_i \cdot x^i$.
Output : True or False. If the algorithm returns True then the
         disk $\Delta_r(m)$ contains exactly $l$ roots.
  Define $E_\ell := |a_l|$ and $E_r := \frac{65}{64} \cdot \sum_{i\neq l}^{i\leq k^2} |a_i|$.
  Define predicate $\mathcal{P} = (E_\ell - E_r > 0)$.
  return output of Algorithm 2 on predicate $\mathcal{P}$ with $\delta = \frac{1}{128}$.
```

Lemma 6. *For a disk $\Delta := \Delta_r(m) \subset \mathbb{C}$, the $\tilde{T}_l$-test needs to compute L-bit approximations of E_ℓ and E_r with $L \leq L(m, r, f) := 2 \cdot (5 + \log n - \log \max_i |a_i|)$. If $T_l(\Delta, \frac{3}{2}, f)$ succeeds, then the $\tilde{T}_l$-test returns True. Running Algorithm 3 for all $l = 0, \ldots, k$ uses a number of bit operations upper bounded by $\tilde{O}(k^2 \cdot (k + \log n)(L(m, r, f) + \tau + n \log \max(1, m) + k^2 \cdot (\log n + \log \max(1, r))))$.*

Proof. From the assumption, it follows that

$$\max_{i=0,\ldots,n} |a_i| = \max_{i=0,\ldots,k^2} |a_i| \leq \frac{1}{2} \cdot \max(|E_\ell|, |E_r|).$$

This yields the claimed bound on the absolute error to which E_ℓ and E_r need to be computed. We now prove correctness. If the algorithm returns True, then $E_\ell > E_r$, and thus $|a_l| > \frac{65}{64} \cdot \sum_{i \neq l}^{i \leq k^2} |a_i|$. If $l = 0$, then $\sum_{i \neq 0} |a_i| < \frac{64}{65} \cdot |a_0| + \frac{1}{128} \cdot |a_0| < |a_0|$. Otherwise, we have $|a_l| > \frac{65}{64} \cdot \sum_{i \neq l}^{i \leq k^2} |a_i| \geq \sum_{i \neq l}^{i \leq k^2} |a_i| + \frac{1}{64} \cdot |a_0| \geq \sum_{i \neq l}^{i \leq n} |a_i|$. Hence, in both cases, $T_l(\Delta, 1, f)$ succeeds, which implies that Δ contains exactly l roots.

Now, suppose that $T_l(\Delta, \frac{3}{2}, f)$ succeeds. If the $\tilde{T}_l$-test returns Undecided, then $\frac{128}{129} \cdot E_\ell < E_r \leq \frac{129}{128} \cdot E_\ell$. On the other hand, we have $|a_l| > \frac{3}{2} \sum_{i \neq l}^{\leq n} |a_i| \geq \frac{3}{2} \sum_{i \neq l}^{\leq k^2} |a_i|$, and thus $E_\ell > \frac{3}{2} E_r$, which contradicts the fact that $\frac{128}{129} \cdot E_\ell < E_r$. If the $\tilde{T}_l$-test returns False, a similar argument yields a contradiction as well. This shows that success of T_l implies that $\tilde{T}_l$ returns True. It remains to show the claimed bounds on the bit complexity. It suffices to estimate the cost for computing an $L(m, r, f)$-bit approximations of E_ℓ and E_r. The i-th coefficient a_i, with $i \leq k^2$, can be computed by evaluating the $(n, k, \tau + k^2 \cdot (\log n + \log \max(1, r)))$-nomial $g_i = f^{(i)}(x) r^i / i!$ at $x = m$. In order to compute $L(m, r, f)$-bit approximations of E_ℓ and E_r, we need to compute an $(L(m, r, f) + 2 \log k)$-bit approximation of each $g_i(m)$, for $i = 0, \ldots, k$. According to Lemma 1, this can be done using $\tilde{O}(k^2 \cdot (k + \log n)(L(m, r, f) + n \log \max(1, m) + \tau + k^2 \cdot (\log n + \log \max(1, r)))$ bit operations. □

Notice that, in order to actually use the $\tilde{T}_l$-test for counting the roots in a disk Δ, we need two conditions to be satisfied. First, we need the condition $\sum_{i > k^2} |a_i| \leq \frac{|a_0|}{128}$ to be true. Second, we need to satisfy the preconditions of the T_l-test.

Theorem 15. *Let f be a (n, k, τ)-nomial as in (1.1), let $\Delta := \Delta_r(m)$ be a disk centered at some $m \in \mathbb{R}_{>0}$ with $\frac{m}{r} > n^{16}$, and let $f_\Delta(x) = \sum_{i=0}^{n} a_i \cdot x^i$. Further suppose that $\Delta_{\frac{r}{k^{4k+2}}}(m)$ does not contain any roots of f. Then, it holds that $\sum_{i > k^2} |a_i| \leq \frac{|a_0|}{128}$.*

Proof. Let $z_1, z_2, \ldots, z_n$ be the complex roots of $F(x)$, then $\frac{a_i}{a_0} = \frac{F^{(i)}(m)}{F(m) \cdot i!} \cdot r^i = \frac{r^i}{i!} \cdot \sum_{(j_1, j_2, \ldots, j_i)} \frac{1}{\prod_{\ell=1}^{i} (m - z_{j_\ell})}$, where we sum over all tuples $(j_1, j_2, \ldots, j_i)$ with distinct entries j_s, $1 \leq j_s \leq n$. For a fixed tuple $(j_1, j_2, \ldots, j_i)$, at most k of the i roots $z_{j_1}, z_{j_2}, \ldots, z_{j_i}$ can appear in the corresponding term of the above sum. At most k of these roots are contained in the code C_n as defined in Figure 2.1, whereas the remaining $i - k$ roots are located outside of C_n. Since $\frac{m}{r} > n^{16}$, the distance from m to any of these roots is at least $n^{15} r$. Also, since $\Delta_{\frac{r}{k^{4k}}}(m)$ does not contain any roots of $F(x)$, distance of m from the roots in C_n is at least $\frac{r}{k^{4k}}$. Thus, we get $\sum_{(j_1, j_2, \ldots, j_i)} \frac{1}{\prod_{\ell=1}^{i} |m - z_{j_k}|} \leq \binom{n}{i} \cdot \frac{k^{4k^2}}{r^k \cdot (n^5 r)^{i-k}}$. Hence, for $i > k^2$, we get

$$\frac{|a_i|}{|a_0|} \leq \frac{r^i}{i!} \cdot \binom{n}{i} \cdot \frac{k^{4k^2+2k}}{r^k \cdot (n^{15} r)^{i-k}} = \frac{1}{i!} \cdot \binom{n}{i} \cdot \frac{k^{4k^2+2k}}{n^{15(i-k)}}$$

$$\leq \frac{1}{i! \cdot i!} \cdot \frac{k^{4k^2+2k}}{n^{14i-15k}} \leq \frac{1}{i! \cdot i!} \cdot \frac{k^{4k^2+2k}}{n^{6i}}$$

(By using the fact that $\binom{n}{i} \leq \frac{n^i}{i!}$ and $15k < 8k^2 < 8i$)

$$\leq \frac{1}{5! \cdot n \cdot i!} \cdot \frac{k^{4k^2+2k}}{k^{6k^2}} \leq \frac{1}{120 \cdot n \cdot i!} \cdot \left(\frac{1}{k^2}\right)^{k^2-2k} < \frac{1}{128n}$$

Hence, summing up over all $i > k^2$ proves the claim. □

The following Corollary is now an immediate consequence of the above theorem and Lemma 15.

Corollary 16. *Let $f(x) \in \mathbb{R}[x]$ be as in (n, k, τ)-nomial as in (1.1). Let $m, r \in \mathbb{R}^+$. Let m^* be a $(M_{D_f}, m[k^2; \frac{r}{k^2}])$ -admissible point and $r^* = 2r$. Define $\Delta = \Delta_{r^*}(m^*) \supseteq \Delta_r(m)$ and $f_\Delta(x) = \sum_{i=0}^{n} a_i \cdot x^i$. Further assume that $\frac{m}{r} \geq 2(1 + n^{16})$, then $\sum_{i > k^2} |a_i| \leq \frac{|a_0|}{128}$.*

In the next step, we show how to satisfy the precondition of the T_l-test. Theorem 14 says that if $\Delta_{256 n^5 r}(m)$ does not contain any of the roots which are not contained in $\Delta_r(m)$, then $T_l(\Delta_{16nr}, f)$ succeeds for some l. Let us define $M = 256 n^5 r$, and let $\Delta_i := \Delta_{M^i r}(m)$ for $i = 0, 1, \ldots, k + 1$. Further assume that r has been chosen sufficiently small enough such that each of disks is contained in the cone C_n. Since C_n contains at most k roots, there must exist a j with $0 \leq j \leq k$ such that $\Delta_{j+1} - \Delta_j$ does not contain any root. Hence the T_l-test will succeed on $\Delta_{16 n M^j r}(m)$. So instead of running the T_l-test on some initial disk $\Delta_r(m)$, we run it on all disks $\Delta_{16 n M^i r}(m)$ for $i = 0, 1, \ldots, k$, and return the first disk on which the T_l-test succeeds; see Algorithm 4.

Correctness of the algorithm follows immediately from the above considerations. The condition on m and r guarantees that each of the disks Δ_i is contained in C_n. Lemma 7 gives a bound on its running time.

Lemma 7. *Algorithm 4 returns a disk $\Delta_{r'}(m')$, with $r' \leq Rr$ and $m - r \leq m' \leq m + r$, together with the number of roots of $f(x)$ in $\Delta_{r'}(m')$. Its bit complexity is bounded by $\tilde{O}(k^5 \cdot (k + \log n) \cdot (k^2 \log n + n \log \max(1, |m|) + \tau + \log \frac{1}{r}))$.*

7 COMPUTING A COVERING

We now show to compute an $(L, [0, 1 + 1/n])$-covering from a weak $(L', [0, 1 + \frac{1}{n}])$-covering, For this, we apply Algorithm 4 to the one-circle regions of the intervals in the weak covering. The following Lemma shows that the requirements in Algorithm 4 are fulfilled if we choose L' large enough. In addition, by ensuring that the intervals in the weak covering are well separated from each other,

Algorithm 4 Wrapper $\tilde{T}_l$-test

```
Input  : A $(n, k, \tau)$-nomial $f(x)$, a disk $\Delta := \Delta_r(m)$ in the
         complex space. We assume $m \geq r + 2Rnr$ with
         $R = 2^{8k+4}n^{5k+16}$.
Output : A disk $\Delta_{r'}(m')$ such that $\Delta_r(m) \subseteq \Delta_{r'}(m')$ along
         with number of roots of $f(x)$ contained in $\Delta_{r'}(m')$
(1) Compute an $(M_{D_f}, m[k^2; \frac{r}{k^2}])$-admissible point $m^*$.
(2) Let $m' = m^*$ and $r' = 2r$.
(3) Let $M = 256n^5r'$.
for each $0 \leq i \leq k$ do
    for each $0 \leq l \leq k$ do
        Perform the $\tilde{T}_l$-test, that is Algorithm 3, on $\Delta_{16nM^i r'}(m')$.
        if $\tilde{T}_l$-test succeeded in the previous step then
            return $\Delta_{16nM^i r'}(m')$ and $l$.
```

Algorithm 5 Computing a $(L, [0, 1 + \frac{1}{n}])$-covering

```
Input  : An $(n, k, \tau)$-nomial $f(x)$ and a positive integer $L$.
Output : An $(L, [0, 1 + 1/n])$-covering for $f$.
(1) Let $R := 2^{8k+4}n^{5k+16}$ and $L' = L + \lceil \log R \rceil + 4\tau + 5$. Compute
    a weak $(L', [0, 1 + \frac{1}{n}])$-covering $\mathcal{L}$ for $f$ that is $(L', 8R)$-separated.
(2) $\mathcal{L}' = \emptyset$
for each interval $I = (a, b) \in \mathcal{L}$ do
    (1) $\Delta = \Delta_{\frac{b-a}{2}}(\frac{a+b}{2})$=one circle region of $I$.
    (2) $(\Delta_{r'}(m'), \mu)$= output of Algo. 4 on $f$ and $\Delta$.
    (3) $\mathcal{L}' = \mathcal{L}' \cup \{(\Delta_{r'}(m'), \mu)\}$
return $\mathcal{L}'$.
```

we can ensure that the corresponding disks returned by Algorithm 4 are disjoint.

Lemma 8. *Algorithm 5 computes an $(L, [0, 1 + \frac{1}{n}])$-covering $\mathcal{L}'$ for f using $\tilde{O}(k^7 \cdot (k + \log n)(k^3 \log n + \tau + L))$ bit operations. The distance between any two disks of $\mathcal{L}'$ is at least $32 \cdot 2^{-L}$, and $\Delta \cap \mathbb{R} \subset (2^{-3\tau}, 2)$ for any disk Δ in $\mathcal{L}'$.*

It remains to show how to compute an $(L, [0, \infty))$-covering for f from an $(L, [0, 1 + \frac{1}{n}))$-covering $\mathcal{L}_1$ for f and an $(L, [0, 1 + \frac{1}{n}))$-covering $\mathcal{L}_2$ for $x^n f(\frac{1}{x})$. We first derive an $(L, [\frac{n}{n+1}, \infty))$-covering for f from $\mathcal{L}_2$ by inverting the disks Δ in $\mathcal{L}_2$. The proof of the following lemma is straight forward.

Lemma 9. *Let $\mathcal{L}$ be an $(L, [0, 1 + \frac{1}{n}])$-covering of $x^n f(\frac{1}{x})$ as computed by Algorithm 5, and $\mathcal{L}' := \{(\Delta^{-1}, \mu) : (\Delta, \mu) \in \mathcal{L}\}$ be the list obtained from $\mathcal{L}$ by inverting the disks in $\mathcal{L}$ (i.e. $\Delta_r(m)^{-1} = \Delta_{r'}(m')$ with $r' = \frac{2r}{m^2 - r^2}$ and $m' = \frac{m}{m^2 - r^2}$). Then, $\mathcal{L}'$ is an $(L', [\frac{n}{n+1}, \infty))$-covering of f with $L' \geq L - 6\tau$ and the distance between two disks in L' is at least $8 \cdot 2^{-L}$.*

Finally, we merge an $(L, [0, 1+1/n))$-covering $\mathcal{L}_1$ and an $(L, [\frac{n}{n+1}, \infty))$-covering $\mathcal{L}_2$ for f. Here, we assume that $L > 3 + \log n$, and that the coverings are computed using Algorithm 5 and by inverting the $(L, (0, 1 + 1/n))$-covering for $x^n \cdot f(1/x)$ to obtain $\mathcal{L}_2$. This guarantees that the distance between any two disks in either $\mathcal{L}_1$ or $\mathcal{L}_2$ is at least $8 \cdot 2^{-L}$. For the merging, we keep each disk from $\mathcal{L}_1$ that has no intersection with a disk from $\mathcal{L}_1$, and vice versa. For each pair of elements $(\Delta_1, \mu_1) \in \mathcal{L}_1$ and $(\Delta_2, \mu_2) \in \mathcal{L}_2$ with $\Delta_1 \cap \Delta_2 \neq \emptyset$, we keep (Δ_1, μ_1) (and omit (Δ_2, μ_2)) if the center of Δ_1 is not larger than 1. Otherwise, we keep (Δ_2, μ_2) (and omit (Δ_1, μ_1)). Following this approach, we might loose some of the complex roots that are contained in the union of Δ_1 and Δ_2, however, we will not loose any real root. Thus, the so obtained list constitutes an $(L, (0, \infty))$-covering for f.
Notice that any two $(L, (0, \infty))$- and $(L, (-\infty, 0))$-coverings for f can be trivially merged by taking their union. In addition, since the final covering contains a list of disjoint disks contained in the union of the cone C_n and its reflection on the imaginary axis, and since the union of these two cones contains at most $2k - 1$ roots of f, the number of disks is also bounded by $2k - 1$. Hence, our main Theorem 3 follows.

REFERENCES

[1] Maria Emilia Alonso Garçia and André Galligo. A root isolation algorithm for sparse univariate polynomials. In *ISSAC*, pages 35–42, 2012.

[2] Osbert Bastani, Christopher J. Hillar, Dimitar Popov, and J. Maurice Rojas. Randomization, Sums of Squares, Near-Circuits, and Faster Real Root Counting. *Contemp. Mathematics*, 556:145–166, 2011.

[3] Ruben Becker, Michael Sagraloff, Vikram Sharma, and Chee-Keng Yap. A near-optimal subdivision algorithm for complex root isolation based on the pellet test and newton iteration. *J. Symb. Comput.*, 2017. In press.

[4] George E. Collins and Rüdiger Loos. Polynomial real root isolation by differentiation. In *SYMSAC*, pages 15–25, 1976.

[5] Michel Coste, Tomás Lajous-Loaeza, Henri Lombardi, and Marie-Francoise Roy. Generalized Budan-Fourier theorem and virtual roots. *J. Complexity*, 21(4):479 – 486, 2005.

[6] F. Cucker, P. Koiran, and S. Smale. A polynomial time algorithm for diophantine equations in one variable. *J. Symb. Comput.*, 27(1):21 – 29, 1999.

[7] Gorav Jindal and Michael Sagraloff. Efficiently Computing Real Roots of Sparse Polynomials . *CoRR*, arXiv:1704.06979, 2017.

[8] Michael Kerber and Michael Sagraloff. Root refinement for real polynomials using quadratic interval refinement. *Journal of Computational and Applied Mathematics*, 280:377 – 395, 2015.

[9] Alexander Kobel, Fabrice Rouillier, and Michael Sagraloff. Computing real roots of real polynomials ... and now for real! In *ISSAC*, pages 303–310, 2016.

[10] Hendrik W. Lenstra (Jr.). Finding small degree factors of lacunary polynomials. *Number Theory in Progress*, 1:267–276, 1999.

[11] J.M. McNamee and Victor Y. Pan. *Numerical Methods for Roots of Polynomials*. Number 2 in Studies in Computational Mathematics. Elsevier Science, 2013.

[12] K. Mehlhorn, M Sagraloff, and P. Wang. From Approximate Factorization to Root Isolation with Application to Cylindrical Algebraic Decomposition. *J. Symb. Comput.*, 66(1):34 – 69, 2015.

[13] V. Pan. Univariate Polynomials: Nearly Optimal Algorithms for Numerical Factorization and Root Finding. *J. Symb. Comput.*, 33(5):701–733, 2002.

[14] Victor Y. Pan, Brian Murphy, Rhys Eric Rosholt, Guoliang Qian, and Yuqing Tang. Real root-finding. In *SNC*, pages 161–169, 2007.

[15] Victor Y. Pan and Elias P. Tsigaridas. On the boolean complexity of real root refinement. In *ISSAC*, pages 299–306, 2013.

[16] J. Maurice Rojas and Yinyu Ye. On solving univariate sparse polynomials in logarithmic time. *J. Complexity*, 21(1):87–110, 2005.

[17] Michael Sagraloff. A near-optimal algorithm for computing real roots of sparse polynomials. In *ISSAC*, pages 359–366, 2014.

[18] Michael Sagraloff and Kurt Mehlhorn. Computing real roots of real polynomials. *Journal of Symbolic Computation*, 73:46 – 86, 2016.

[19] C.K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, 2000.

[20] Yinyu Ye. Combining Binary Search and Newton's Method to Compute Real Roots for a Class of Real Functions. *J. Complexity*, 10(3):271 – 280, 1994.