Complexity Analysis of Root Clustering for a Complex Polynomial

Ruben Becker[*]
MPI for Informatics,
Saarbrücken Graduate School
of Computer Science
Saarbrücken, Germany
ruben@mpi-inf.mpg.de

Michael Sagraloff[†]
MPI for Informatics
Saarbrücken, Germany
msagralo@mpi-inf.mpg.de

Vikram Sharma[†]
Institute of Mathematical
Sciences
Chennai, India
vikram@imsc.res.in

Juan Xu[‡]
Courant Institute of
Mathematical Sciences
NYU, New York, USA
jx732@nyu.edu

Chee Yap[§]
Courant Institute of
Mathematical Sciences
NYU, New York, USA
yap@cs.nyu.edu

ABSTRACT

Let $F(z)$ be an arbitrary complex polynomial. We introduce the **local root clustering problem**, to compute a set of natural ε-clusters of roots of $F(z)$ in some box region B_0 in the complex plane. This may be viewed as an extension of the classical root isolation problem. Our contribution is two-fold: we provide an efficient certified subdivision algorithm for this problem, and we provide a bit-complexity analysis based on the local geometry of the root clusters.

Our computational model assumes that arbitrarily good approximations of the coefficients of F are provided by means of an oracle at the cost of reading the coefficients. Our algorithmic techniques come from a companion paper [3] and are based on the Pellet test, Graeffe and Newton iterations, and are independent of Schönhage's splitting circle method. Our algorithm is relatively simple and promises to be efficient in practice.

1. INTRODUCTION

The problem of computing the roots of a univariate polynomial F has a venerable history that dates back to antiquity. With the advent of modern computing, the subject received several newfound aspects [17, 20]; in particular, the introduction of algorithmic rigor and complexity analysis has been extremely fruitful. This development is usually traced to Schönhage's 1982 landmark paper, "*Fundamental Theorem of Algebra in Terms of Computational Complexity*" [28]. Algorithms in this tradition are usually described as "exact and efficient". Schönhage considers the problem of approximate polynomial factorization, that is, the computation of approximations $\tilde{z}_i$ of the roots z_i of F such that $\|F - \widetilde{F}\|_1 < 2^{-b} \cdot \|F\|_1$, where $\widetilde{F}(z) := \mathrm{lcf}(F) \cdot \prod_{i=1}^{n}(z - \widetilde{z}_i)$ and b is a given positive integer. The sharpest result for this problem is given by Pan [22, Theorem 2.1.1], [20, p.196]. Hereafter, we refer to the underlying algorithm in this theorem as "Pan's algorithm". Under some mild assumption on F (i.e., $|z_i| \leq 1$ and $b \geq n \log n$), Pan's algorithm uses only $\tilde{O}(n \log b)$ arithmetic operations with a precision bounded by $\tilde{O}(b)$, and thus $\tilde{O}(nb)$ bit operations. This result further implies that the complexity of approximating all z_i's to any specified b/n bits, with $b > n \log n$, is also $\widetilde{O}(nb)$ [22, Corollary 2.1.2]. Here, $\widetilde{O}$ means we ignore logarithmic factors in the displayed parameters. In a model of computation, where it is assumed that the coefficients of F are complex numbers for which approximations are given up to a demanded precision, the above bound is tight (up to polylogarithmic factors) for polynomial factorization as well as for root approximation.

In parallel, a major focus of exact and efficient root approximation research has been to determine the complexity of *isolating* all the roots of an *integer* polynomial $F(z)$ of degree n with L-bit coefficients. We call this the **benchmark problem** [27] since this case is the main theoretical tool for comparing root isolation algorithms. Although this paper addresses complex root isolation, we will also refer to the related **real benchmark problem** which concerns real roots for integer polynomials.

The problem of isolating the roots of a polynomial can be reduced to approximate polynomial factorization. Schönhage showed that, for a square-free polynomial, it suffices to choose a b of size $\Omega(n(\log n + L))$ to ensure that the distance between the approximations $\tilde{z}_i$ and the actual roots z_i is small enough to directly deduce isolating regions of

[*]Supported by the Max Planck Center for Visual Computing and Communication.
[†]Supported by IMPECS (Indo-German Max Planck Center for Computer Science).
[‡]Supported by China Scholarship Council, No.20150602005.
[§]Supported by NSF Grant #CCF-1423228.

ISSAC '16, July 19-22, 2016, Waterloo, ON, Canada
 ISBN 978-1-4503-4380-0/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2930889.2930939

the z_i's. Together with Pan's result on approximate polynomial factorization, this yields a complexity of $\widetilde{O}(n^2L)$ for the benchmark problem. Interestingly, the latter bound was not explicitly stated until recently ([11, Theorem 3.1]).

Mehlhorn et al. [18] extend the latter result to (not necessarily square-free) polynomials F with arbitrary complex coefficients for which the number of distinct roots is given as an additional input. That is, Pan's algorithm is used as a blackbox with successively increasing precision b to isolate the roots of F. For the benchmark problem, this yields the bound $\widetilde{O}(n^3 + n^2L)$; however, the actual cost adapts to the geometry of the roots, and for most input polynomials, the complexity is considerably lower than the worst case bound.

We further remark that it seems likely that the bound $\widetilde{O}(n^2L)$ is also near-optimal for the benchmark problem because it is generally believed that Pan's algorithm is near-optimal for the problem of approximately factorizing a polynomial with complex coefficients. However, rigorous arguments for such claims are missing.

It had been widely assumed that near-optimal bounds need the kind of "muscular" divide and conquer techniques such as the splitting circle method of Schönhage (which underlies Pan's algorithm and most of the fast algorithms in the complexity literature). These algorithms are far from practical (see below). So, also the bound $\widetilde{O}(n^2(n+L))$ achieved by Mehlhorn et al. [18] is mainly of theoretical interest as the algorithm uses Pan's method as a blackbox.

This paper is interested in subdivision methods. The classical example here is root isolation based on Sturm sequences. Two types of subdivision algorithms are actively investigated currently: the **Descartes Method** [7, 15, 24, 28, 25, 26] and the **Evaluation Method** [5, 4, 29, 2, 27, 13, 21]. See [27] for a comparison of Descartes and Evaluation (or Bolzano) methods.

The development of certain tools, such as the Mahler-Davenport root bounds [8, 9], have been useful in deriving tight bounds on the subdivision tree size for certain subdivision algorithms [10, 4, 29]. Moreover, most of these analyses can be unified under the "continuous amortization" framework [5, 6] which can even incorporate bit-complexity. However, these algorithms only use bisection in their subdivision, which seems destined to lag behind the above "near optimal bounds" by a factor of n. To overcome this, we need to combine Newton iteration with bisection, an old idea that goes back to Dekker and Brent in the 1960s. In recent years, a formulation of Newton iteration due to Abbott [1] and Sagraloff [25] has proven especially useful. This has been adapted to achieve the recent near-optimal algorithms of Sagraloff and Mehlhorn [25, 26] for real roots, and [3] for complex roots.

The Root Clustering Problem. In this paper, we are interested in root clustering. The requirements of root clustering represents a simultaneous strengthening of root approximation (i.e., the output discs must be disjoint) and weakening of root isolation (i.e., the output discs can have more than one root). Hereafter, "root finding" refers generally to any of the tasks of approximating, isolating or clustering roots.

For an analytic function $F : \mathbb{C} \to \mathbb{C}$ and a complex disc $\Delta \subseteq \mathbb{C}$, let $\mathcal{Z}(\Delta; F)$ denote the multiset of roots of F in Δ and $\#(\Delta; F)$ counts the size of this multiset. We write $\mathcal{Z}(\Delta)$ and $\#(\Delta)$ since F is usually supplied by the context. Any non-empty set of roots of the form $\mathcal{Z}(\Delta)$ is called a **cluster**. The disc Δ is called an **isolator** for F if $\#(\Delta) = \#(3\Delta) > 0$. Here, $k\Delta = k \cdot \Delta$ denotes the centrally scaled version of Δ by a factor $k \geq 0$. The set $\mathcal{Z}(\Delta)$ is called a **natural cluster** when Δ is an isolator. A set of n roots could contain $\Theta(n^3)$ clusters, but at most $2n - 1$ of these are natural. This follows from the fact that any two natural clusters are either disjoint or have a containment relationship. The benchmark problem is a global problem because it concerns *all* roots of the polynomial $F(z)$; we now address local problems where we are interested in finding only *some* roots of $F(z)$. For instance, Yakoubson [30] gave a method to test if Newton iteration from a given point will converge to a cluster. In [31], we introduced the following **local root clustering problem**: *given $F(z)$, a box $B_0 \subseteq \mathbb{C}$ and $\varepsilon > 0$, to compute a set $\{(\Delta_i, m_i) : i \in I\}$ where the Δ_i's are pairwise disjoint isolators, each of radius $\leq \varepsilon$ and $m_i = \#(\Delta_i) \geq 1$, such that*

$$\mathcal{Z}(B_0) \;\subseteq\; \bigcup\nolimits_{i \in I} \mathcal{Z}(\Delta_i) \;\subseteq\; \mathcal{Z}(2B_0).$$

We call the set $S = \{\Delta_i : i \in I\}$ (omitting the m_i's) a **solution** for the local root clustering instance $(F(z), B_0, \varepsilon)$. The roots in $2B_0 \setminus B_0$ are said to be **adventitious** because we are really only interested in roots in B_0. Suppose S and $\widehat{S}$ are both solutions for an instance $(F(z), B_0, \varepsilon)$. If $S \subseteq \widehat{S}$, then we call $\widehat{S}$ an augmentation of S. Thus any $\Delta \in \widehat{S} \setminus S$ contains only adventitious roots.

We solved the local root clustering problem in [31] for any analytic function F, provided an upper on $\#(2B_0)$ is known, but no complexity analysis was given. Let us see why our formulation is reasonable. It is easy to modify our algorithm so that the adventitious roots in the output are contained in $(1 + \delta)B_0$ for any fixed $\delta > 0$. We choose $\delta = 1$ for convenience. Some $\delta > 0$ is necessary because in our computational model where only approximate coefficients of F are available, we cannot decide the implicit "Zero Problem" [33] necessary to decide if the input has a root on the boundary of B_0, or to decide whether Δ contains a root of multiplicity $k > 1$. Thus, root clustering is the best one can hope for.

1.1 Main Result

In this paper, we describe a local root clustering algorithm and provide an analysis of its bit-complexity. Standard complexity bounds for root isolation are based on **synthetic parameters** such as degree n and bitsize L of the input polynomial. But our computational model for $F(z)$ has no notion of bit size. Moreover, to address "local" complexity of roots, we must invoke **geometric parameters** such as root separation [25, 26]. We will now introduce new geometric parameters arising from cluster considerations.

Assume $F(z)$ has m distinct complex roots $z_1, \ldots, z_m$ where z_j has multiplicity $n_j \geq 1$, the degree of $F(z)$ is $n = \sum_{j=1}^{m} n_j$, and the leading coefficient of F has magnitude $\geq 1/4$. Let k be the number of roots counted with multiplicities in $2B_0$. An input instance $(F(z), B_0, \varepsilon)$ is called **normal** if $k \geq 1$ and $\varepsilon \leq \min\left\{1, \frac{w_0}{96n}\right\}$ with w_0 the width of B_0. For any set $U \subseteq \mathbb{C}$, let $\overline{\log}(U) := \max(1, \log \sup(|z| : z \in U))$.

Our algorithm outputs a set of discs, each one contains a natural cluster. We provide a bit complexity bound of the algorithm in terms of the output.

Theorem A *Let S be the solution computed by our algorithm for a normal instance $(F(z), B_0, \varepsilon)$. Then there is an augmentation $\widehat{S} = \{D_i : i \in I\}$ of S such that the bit*

complexity of the algorithm is

$$\widetilde{O}\Big(n^2 \,\overline{\log}(B_0) + n\sum\nolimits_{D\in\widehat{S}} L_D\Big) \tag{1}$$

with

$$\begin{aligned} L_D =& \widetilde{O}\Big(\tau_F + n\cdot\overline{\log}(\xi_D) + k_D\cdot(k+\overline{\log}(\varepsilon^{-1})) \\ &+ \overline{\log}(\prod\nolimits_{z_j\notin D}|\xi_D - z_j|^{-n_j})\Big) \end{aligned} \tag{2}$$

where $k_D = \#(D)$, and ξ_D is an arbitrary root in D. Moreover, an L_D^*-bit approximation of the coefficients of F is required with $L_D^* := \max_{D\in\widehat{S}} L_D$.

The solution $\widehat{S}$ in this theorem is called the **augmented solution** for input $(F(z), B_0, \varepsilon)$. Each natural ε-cluster $D \in \widehat{S}$ is an isolator of radius $\leq \varepsilon$. From (1), we deduce:

Corollary to Theorem A
The bit complexity of the algorithm is bounded by

$$\widetilde{O}\Big(n^2(\tau_F + k + m) + nk\,\overline{\log}(\varepsilon^{-1}) + n\,\overline{\log}\,|\,\mathrm{GenDisc}(F_\varepsilon)|^{-1}\Big). \tag{3}$$

In case F is an integer polynomial, this bound becomes

$$\widetilde{O}\Big(n^2(\tau_F + k + m) + nk\,\overline{\log}(\varepsilon^{-1})\Big). \tag{4}$$

The bound (4) is the sum of two terms: the first is essentially the near-optimal root bound, the second is linear in k, n and $\overline{\log}(\varepsilon^{-1})$. This suggests that Theorem A is quite sharp.

On strong ε-clusters. Actually, the natural ε-clusters in the $\widehat{S}$ have some intrinsic property captured by the following definition. Two roots z, z' of F are **ε-equivalent**, written $z \overset{\varepsilon}{\sim} z'$, if there exists a disk $\Delta = \Delta(r, m)$ containing z and z' such that $r \leq \frac{\varepsilon}{12}$ and $\#(\Delta) = \#(114\cdot\Delta)$. Clearly Δ is an isolator; from this, we see that ε-equivalence is an equivalence relationship. We define a **strong ε-cluster** to be any such ε-equivalence class. Unlike natural clusters, any two strong ε-clusters must be disjoint.

Theorem B
Each natural cluster $D \in \widehat{S}$ is a union of strong ε-clusters.

This implies that our algorithm will never split any strong ε-cluster. It might appear surprising that our "soft" techniques can avoid accidentally splitting a strong ε-cluster.

1.2 What is New

Our algorithm and analysis is noteworthy for its wide applicability: (1) We do not require square-free polynomials. This is important because we cannot compute the square-free part of $F(z)$ in our computational model where the coefficients of $F(z)$ are only arbitrarily approximated. Most of the recent fast subdivision algorithms for real roots [25, 26] require square-free polynomials. (2) We address the local root problem and provide a complexity analysis based on the local geometry of roots. Many practical applications (e.g., computational geometry) can exploit locality. The companion paper [3] also gives a local analysis. However, it is under the condition that the initial box is not too large or is centered at the origin, and an additional preprocessing step is needed for the latter case. But our result does not depend on any assumptions on B_0 nor require any preprocessing. (3) Our complexity bound is based on cluster geometry instead of individual roots. To see its benefits, recall that the bit complexity in [3] involves a term $\overline{\log}\,\sigma(z_i)^{-1}$ where $\sigma(z_i)$ is the distance to the nearest root of $F(z)$. If z_i is a multiple root, $\sigma(z_i) = 0$. If square-freeness is not assumed, we must replace $\sigma(z_i)$ by the distance $\sigma^*(z_i)$ to the closest root $\neq z_i$ (so $\sigma^*(z_i) > 0$). But in fact, our bound in (1) involves $T_D := \overline{\log}\prod_{z_j\notin D}|\xi_i - z_j|^{-n_j}$ which depends only on the inverse distance from a root within a cluster D to the other roots outside of D, which is smaller than $\overline{\log}\,\sigma^*(z_i)^{-1}$. So the closeness of roots within D has no consequence on T_D.

Why can't we just run the algorithm in [3] by changing the stopping criteria so that it terminates as soon as a component C is verified to be a natural ε-cluster? Yes, indeed one can. But our previous method of charging the work associated with a box B to a root $\phi(B)$ may now cause a cluster of multiplicity k to be charged a total of $\Omega(k)$ times, instead of $\widetilde{O}(1)$ times. Cf. Lemma 11 below where $\phi(B)$ is directly charged to a cluster.

1.3 Practical Significance

Our algorithm is not only theoretically efficient, but has many potential applications. Local root isolation is useful in applications where the roots of interest lie in a known locality, and this local complexity can be much smaller than that of finding all roots. From this perspective, focusing on the benchmark problem is misleading for such applications.

We believe our algorithm is practical, and plan to implement it. Many recent subdivision algorithms were implemented, with promising results: Rouillier and Zimmermann [24] engineered a very efficient Descartes method algorithm which is widely used in the Computer Algebra community, through Maple. The `CEVAL` algorithm in [27] was implemented in [12, 13]. Kobel, Rouillier and Sagraloff[1] implemented the `ANewDsc` algorithm from [26]. Becker [2] gave a Maple implementation of the `REVAL` algorithm for isolating real roots of a square-free real polynomial.

In contrast, none of the divide-and-conquer algorithms [23, 19, 14] have been implemented. Pan notes [22, p. 703]: "*Our algorithms are quite involved, and their implementation would require a non-trivial work, incorporating numerous known implementation techniques and tricks.*" Further [22, p. 705] "*since Schönhage (1982b) already has 72 pages and Kirrinnis (1998) has 67 pages, this ruled out a self-contained presentation of our root-finding algorithm*". But our paper [3] is self-contained with over 50 pages, and explicit precision requirements for all numerical primitives.

2. PRELIMINARY

We review the basic tools from [3]. The coefficients of F are viewed as an oracle from which we can request approximations to any desired absolute precision. Approximate complex numbers are represented by a pair of dyadic numbers, where the set of dyadic numbers (or BigFloats) may be denoted $\mathbb{Z}[\frac{1}{2}] := \{n2^m : n, m \in \mathbb{Z}\}$. We formalize[2] this as follows: a complex number $z \in \mathbb{C}$ is an **oracular number** if it is represented by an **oracle function** $\widetilde{z} : \mathbb{N} \to \mathbb{Z}[\frac{1}{2}]$ with some $\tau \geq 0$ such that for all $L \in \mathbb{N}$, $|\widetilde{z}(L) - z| \leq 2^{-L}$ and $\widetilde{z}(L)$ has $O(\tau + L)$ bits. The oracular number is said to

[1] Private communication.
[2] This is essentially the "bit-stream model", but the term is unfortunate because it suggests that we are getting successive bits of an infinite binary representation of a real number. We know from Computable Analysis that this representation of real numbers is not robust.

be τ-**regular** in this case. In our computational model, the algorithm is charged the cost to read these $O(\tau + L)$ bits. This cost model is reasonable when z is an algebraic number because in this case, $\widetilde{z}(L)$ can be computed in time $\widetilde{O}(\tau+L)$ on a Turing machine. Following [3, 31], we can construct a procedure $\texttt{SoftCompare}(z_\ell, z_r)$ that takes two non-negative real oracular numbers z_ℓ and z_r with $z_\ell+z_r > 0$, that returns a value in $\{+1, 0, -1\}$ such that if $\texttt{SoftCompare}(z_\ell, z_r)$ returns 0 then $\frac{2}{3}z_\ell < z_r < \frac{3}{2}z_\ell$; otherwise $\texttt{SoftCompare}(z_\ell, z_r)$ returns $\texttt{sign}(z_\ell - z_r) \in \{+1, -1\}$. Note that $\texttt{SoftCompare}$ is non-deterministic since its output depends on the underlying oracular functions used.

LEMMA 1 (see [3, Lemma 4] and [31]).
In evaluating $\texttt{SoftCompare}(z_\ell, z_r)$*:*
(a) The absolute precision requested from the oracular numbers z_ℓ and z_r is at most $L = 2(\overline{\log}(\max(z_\ell, z_r)^{-1}) + 4)$.
(b) The time complexity of the evaluation is $\widetilde{O}(\tau + L)$ where z_ℓ, z_r are τ-regular.

The critical predicate for our algorithm is a test from Pellet (1881) (see [16]). Let $\Delta = \Delta(m, r)$ denote a disc with radius $r > 0$ centered at $m \in \mathbb{C}$. For $k = 0, 1, \ldots, n$ and $K \geq 1$, define the **Pellet test** $T_k(\Delta, K) = T_k(\Delta, K; F)$ as the predicate

$$|F_k(m)|r^k > K \cdot \sum_{i=0, i\neq k}^{n} |F_i(m)|r^i$$

Here $F_i(m)$ is defined as the Taylor coefficient $\frac{F^{(i)}(m)}{i!}$. Call the test $T_k(\Delta, K)$ a **success** if the predicate holds; else a **failure**. Pellet's theorem says that for $K \geq 1$, a success implies $\#(\Delta) = k$. Following [31, 3], we define the "soft version" of Pellet test $\widetilde{T}_k(\Delta)$ to mean that $\texttt{SoftCompare}(z_\ell, z_r) > 0$ where $z_\ell = |F_k(m)|r^k$ and $z_r = \sum_{i=0,i\neq k}^{n} |F_i(m)|r^i$. We need to derive quantitative information in case the soft Pellet test fails. Contra-positively, what quantitative information ensures that the soft Pellet test will succeed? Roughly, it is that $\#(\Delta) = \#(r\Delta) = k$ for a suitably large $r > 1$, as captured by the following theorem:

THEOREM 2.
Let k be an integer with $0 \leq k \leq n = \deg(F)$ and $K \geq 1$.
Let $c_1 = 7kK$, and $\lambda_1 = 3K(n-k) \cdot \max\{1, \{4k(n-k)\}\}$.
If $\#(\Delta) = \#(c_1\lambda_1\Delta) = k$, then

$$T_k(c_1\Delta, K, F) \quad \textit{holds.}$$

The factor $c_1\lambda_1$ is $O(n^4)$ in this theorem, an improvement from $O(n^5)$ in [3]. A proof is given in Appendix A. In application, we choose $K = \frac{3}{2}$ and thus $c_1 \cdot \lambda_1 \leq (7Kn) \cdot (12Kn^3) = 189n^4$. The preceding theorem implies that if $\#(\Delta) = \#(189n^4\Delta)$ then $T_k(\frac{21}{2}n\Delta, \frac{3}{2}, F)$ holds. This translates into the main form for our application:

COROLLARY
If $k = \#(\frac{1}{11}n\Delta) = \#(18n^3\Delta)$ then $T_k(\Delta, \frac{3}{2}; F)$ holds.

In other words, under the hypothesis of this Corollary, $\widetilde{T}_k(\Delta)$ succeeds. We need one final extension: instead of applying $\widetilde{T}_k(\Delta)$ directly on F, we apply $\widetilde{T}_k(\Delta(0,1))$ to the Nth Graeffe iterations of $F_\Delta(z) := F(m + rz)$. Here, $\Delta = \Delta(m, r)$ and $N = \lceil \log(1 + \log n) \rceil + 4 = O(\log\log n)$. The result is called the **Graeffe-Pellet test**, denoted $\widetilde{T}_k^G(\Delta) = \widetilde{T}_k^G(\Delta; F)$. As in [3] we combine $\widetilde{T}_k^G(\Delta)$ for all $k = 0, 1, \ldots, n$ to obtain

$$\widetilde{T}_*^G(\Delta)$$

which returns the unique $k \in \{0, \ldots, n\}$ such that $\widetilde{T}_k^G(\Delta)$ succeeds, or else returns -1. We say that the test $\widetilde{T}_*^G(\Delta)$ **succeeds** iff $T_*^G(\Delta, K) \geq 0$.

The key property of $\widetilde{T}_i^G(\Delta)$ is [3, Lemma 6]:

LEMMA 3 (Soft Graeffe-Pellet Test).
Let $\rho_1 = \frac{2\sqrt{2}}{3} \simeq 0.943$ and $\rho_2 = \frac{4}{3}$.
(a) If $\widetilde{T}_k^G(\Delta)$ succeeds then $\#(\Delta) = k$.
(b) If $\widetilde{T}_*^G(\Delta)$ fails then $\#(\rho_2\Delta) > \#(\rho_1\Delta)$.

The bit complexity of the combined test $\widetilde{T}_*^G(\Delta)$ is asymptotically the same as any individual test [3, Lemma 7]:

LEMMA 4. *Let*

$$L(\Delta, F) := 2 \cdot (4 + \overline{\log}(\|F_\Delta\|_\infty^{-1})).$$

(a) To evaluate $\widetilde{T}_k^G(\Delta)$, it is sufficient to have an M-bit approximation of each coefficient of F where $M = \widetilde{O}(n\,\overline{\log}(m, r) + \tau_F + L(\Delta, F))$.
(b) The total bit-complexity of computing $\widetilde{T}_*^G(\Delta)$ is $\widetilde{O}(nM)$.

2.1 Box Subdivision

Let $A, B \subseteq \mathbb{C}$. Their **separation** is $\mathrm{Sep}(A, B) := \inf\{|a - b| : a \in A, b \in B\}$, and $\mathrm{rad}(A)$, the **radius** of A, is the smallest radius of a disc containing A. Also, ∂A denotes the boundary of A.

We use the terminology of subdivision trees (quadtrees) [3]. All boxes are closed subsets of $\mathbb{C}$ with square shape and axes-aligned. Let $B(m, w')$ denote the axes-aligned box centered at m of width $w(B) := w'$. As for discs, if $k \geq 0$ and $B = B(m, w')$, then kB denotes the box $B(m, kw')$. The smallest covering disc of $B(m, w')$ is $\Delta(m, \frac{1}{\sqrt{2}}w')$. If $B = B(m, w')$ then we define $\Delta(B)$ as the disc $\Delta(m, \frac{3}{4}w')$. Thus $\Delta(m, \frac{1}{\sqrt{2}}w')$ is properly contained in $\Delta(B)$. Any collection $\mathcal{S}$ of boxes is called a (box) **subdivision** if the interior of any two boxes in $\mathcal{S}$ are disjoint. The union $\bigcup\mathcal{S}$ of these boxes is called the **support** of S. Two boxes B, B' are **adjacent** if $B \cup B'$ is a connected set, equivalently, $B \cap B' \neq \emptyset$. A subdivision $\mathcal{S}$ is said to be **connected** if its support is connected. A **component** C is the support of some connected subdivision $\mathcal{S}$, i.e., $C = \bigcup\mathcal{S}$.

The **split** operation on a box B creates a subdivision $\texttt{Split}(B) = \{B_1, \ldots, B_4\}$ of B comprising four congruent subboxes. Each B_i is a **child** of B, denoted $B \to B_i$. Therefore, starting from any box B_0, we may split B_0 and recursively split zero or more of its children. After a finite number of such splits, we obtain a **subdivision tree** rooted at B_0, denoted $\mathcal{T}_{subdiv}(B_0)$.

The **exclusion test** for a box $B(m, w')$ is $\widetilde{T}_0^G(\Delta(m, \frac{3w'}{4})) = \widetilde{T}_0^G(\Delta(B))$. We say that $B(m, w')$ is **excluded** if this test succeeds, and **included** if it fails. The key fact we use is a consequence of Lemma 3 for the test $\widetilde{T}_0^G(\Delta)$:

COROLLARY 5. *Consider any box $B = B(m, w')$.*
(a) If B is excluded, then $\#(\Delta(m, \frac{3w'}{4})) = 0$, so $\#(B) = 0$.
(b) If B is included, then $\#(\Delta(m, w')) > 0$, so $\#(2B) > 0$.

2.2 Component Tree

In traditional subdivision algorithms, we focus on the complexity analysis on the subdivision tree $\mathcal{T}_{subdiv}(B_0)$. But for

our algorithm, it is more natural to work with a tree whose nodes are higher level entities called components above.

Typical of subdivision algorithms, our algorithm consists of several while loops, but for now, we only consider the main loop. This loop is controlled by the **active queue** Q_1. At the start of each loop iteration, there is a set of included boxes. The maximally connected sets in the union of these boxes constitute our (current) components. And the boxes in the subdivision of a component C are called the **constituent boxes** of C. While Q_1 is non-empty, we remove a component C from Q_1 for processing. There are 3 dispositions for C: We try to put C to the **output queue** Q_{out}. Failing this, we try a **Newton Step**. If successful, it produces a single new component $C' \subset C$ which is placed in Q_1. If Newton Step fails, we apply a **Bisection Step**. In this step, we split each constituent box of C, and apply the exclusion test to each of its four children. The set of included children are again organized into maximally connected sets $C_1, \ldots, C_t$ $(t \geq 1)$. Each subcomponent C_i is either placed in Q_1 or Q_{dis}, depending on whether C_i intersects the initial box B_0. The components in Q_{dis} are viewed as **discarded** because we do not process them further (but our analysis need to ensure that other components are sufficiently separated from them in the main loop). We will use the notation $C \to C'$ or $C \to C_i$ to indicate the parent-child relationship. The **component tree** is defined by this parent-child relationship, and denoted $\mathcal{T}_{comp}$. In [3], the root of the component tree is B_0; we take $\frac{5}{4}B_0$ as the root to address boundary issues. So we write $\mathcal{T}_{comp} = \mathcal{T}_{comp}(\frac{5}{4}B_0)$ to indicate that $\frac{5}{4}B_0$ is the root. The leaves of $\mathcal{T}_{comp}$ are either discarded (adventitious) or output.

For efficiency, the set of boxes in the subdivision of a component C must maintain links to adjacent boxes within the subdivision; such links are easy to maintain because all the boxes in a component have the same width.

3. COMPONENT PROPERTIES

Before providing details about the algorithm, we discuss some critical data associated with each component C. Such data is subscripted by C. We also describe some qualitative properties so that the algorithm can be intuitively understood. Figure 1 may be an aid in the following description.

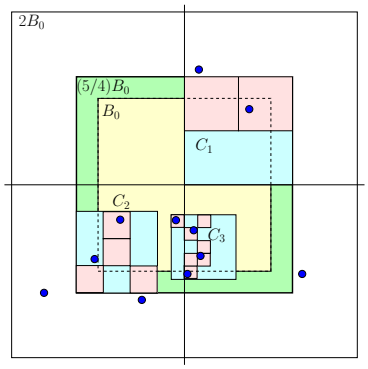

Figure 1: Three components C_1, C_2, C_3: blue dots indicate roots of F, pink boxes are constituent boxes, and the non-pink parts of each B_C is colored cyan. Only C_3 is confined.

(C1) All the constituent boxes of a component share a common width, denoted by w_C.

(C2) Our algorithm never discards any box B if B contains a root in B_0; it follows that all the roots in B_0 are contained in $\bigcup_C C$ where C ranges over components in $Q_0 \cup Q_1 \cup Q_{out}$ (at any moment during our algorithm).

(C3) Recall that a zero ζ of $F(z)$ in $2B_0 \setminus B_0$ is called adventitious. A component C is **adventitious** if $C \cap B_0$ is empty (placed in Q_{dis}). We say a component C is **confined** if $C \cap \partial(\frac{5}{4}B_0)$ is empty; otherwise it is non-confined. Figure 2 shows these different kinds of components. Note that after the preprocessing step, all components are confined.

(C4) If C, C' are distinct active components, then their separation $\mathrm{Sep}(C, C')$ is at least $\max\{w_C, w_{C'}\}$. If C is an adventitious component, then $\mathrm{Sep}(C, B_0) \geq w_C$. If C is a confined component, then $\mathrm{Sep}(C, \partial(\frac{5}{4}B_0)) \geq w_C$.

(C5) Let C^+ be the **extended component** defined as the set $\bigcup_{B \in \mathcal{S}_C} 2B$. If C and C' are distinct components, then C^+ and C'^+ are disjoint. Moreover, if C is confined, then $\#(C) = \#(C^+)$ (see Appendix B).

(C6) Define the **component box** B_C to be any smallest square containing C subject to $B_C \subseteq (5/4)B_0$. Define W_C as the width of B_C and the disc $\Delta_C := \Delta(B_C)$. Define R_C as the radius of Δ_C; note that $R_C = \frac{3}{4}W_C$.

(C7) Each component is associated with a "Newton speed" denoted by N_C with $N_C \geq 4$. A key idea in the Abbott-Sagraloff technique for Newton-Bisection is to automatically update N_C: if Newton fails, the children of C have speed $\max\{4, \sqrt{N_C}\}$ else they have speed N_C^2.

(C8) Let $k_C := \#(\Delta_C)$, the number of roots of $\mathcal{Z}(\Delta_C)$, *counted with multiplicity*. Note that k_C is not always available, but it is needed for the Newton step. We try to determine k_C before the Newton Step in the main loop.

(C9) A component C is **compact** if $W_C \leq 3w_C$. Such components have many nice properties, and we will require output components to be compact.

In recap, each component C is associated with the data:

$$w_C, W_C, M_C, B_C, \Delta_C, R_C, k_C, N_C.$$

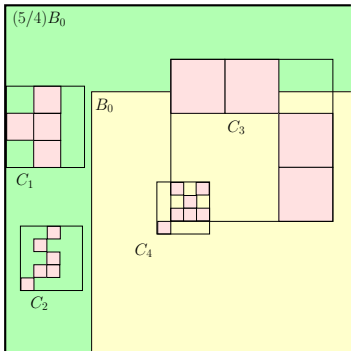

Figure 2: Four types of components: C_1 is not confined, the rest are confined; C_1 and C_2 are adventitious; C_3 may contain adventitious roots; C_4 has no adventitious roots.

4. THE CLUSTERING ALGORITHM

As outlined above, our clustering algorithm is a process for constructing and maintaining components, globally controlled by queues containing components. Each component C represents a non-empty set of roots. In addition to the queues Q_1, Q_{out}, Q_{dis} above, we also need a **preprocessing queue** Q_0. Furthermore, Q_1 is a priority queue such that the operation $C \leftarrow Q_1.pop()$ returns the component with the largest width W_C.

We first provide a high level description of the two main subroutines.

The **Newton Step** $Newton(C)$ is directly taken from [3]. This procedure takes several arguments, $Newton(C, N_C, k_C, x_C)$. The intent is to perform an order k_C Newton step:

$$x'_C \leftarrow x_C - k_C \frac{F(x_C)}{F'(x_C)}.$$

We then check whether $\mathcal{Z}(C)$ is actually contained in the small disc $\Delta' := \Delta(x'_C, r')$ where

$$r' := \max\{\varepsilon, w_C/(8N_C)\}. \tag{5}$$

This amounts to checking whether $\widetilde{T}^G_{k_C}(\Delta')$ succeeds. If it does, Newton test succeeds, and we return a new component C' that contains $\Delta' \cap C$ with speed $N_{C'} := (N_C)^2$ and constituent width $w_{C'} := \frac{w_C}{2N_C}$. The new component C' consist of at most 4 boxes and $W_{C'} \le 2w_{C'}$. In the original paper [3], r' was simply set to $\frac{w_C}{8N_C}$; but (5) ensures that $r' \ge \varepsilon$. This avoids the overshot of Newton Step and simplifies our complexity analysis. If $\widetilde{T}^G_{k_C}(\Delta')$ fails, then Newton test fails, and it returns an empty set. In the following context, we simply denote this routine as "$Newton(C)$".

The **Bisection Step** $Bisect(C)$ returns a set of components. Since it is different from that in [3], we list the modified bisection algorithm in Figure 3.

We list the clustering algorithm in Figure 4.

Remarks on Root Clustering Algorithm:

1. In the preprocessing stage, for each component C, $w_C \ge \frac{w(B_0)}{48n}$ (see Appendix B). Thus depth of C in $\mathcal{T}_{comp}$ is $O(\log n)$.
2. In the main stage, each component C is confined. Moreover, the separation of $\partial((5/4)B_0)$ from $\partial(2B_0)$ is $\frac{3}{8}w(B_0)$. It follows that $2B_C \subseteq 2B_0$ (using the fact that $W_C \le w(B_0)/2$ from preprocessing).
3. The steps in this algorithm should appear well-motivated (after [3]). The only non-obvious step is the test "$W_C \le 3w_C$" (colored in red). This part is only needed for the analysis; the correctness of the algorithm is not impacted if we simply replace this test by the Boolean constant `true` (i.e., allowing the output components to have $W_C > 3w_C$).
4. We ensure that $W_C \ge \varepsilon$ before we attempt to do the Newton Step. This is not essential, but simplifies the analysis.

Based on the stated properties, we prove the correctness of our algorithm (see Appendix B).

THEOREM 6 (Correctness). *The Root Clustering Algorithm halts and outputs a collection $\{(\Delta_C, k_C) : C \in Q_{out}\}$ of pairwise disjoint ε-isolators such that $\mathcal{Z}(B_0) \subseteq \bigcup_{C \in Q_{out}} \mathcal{Z}(\Delta_C) \subseteq \mathcal{Z}(2B_0)$.*

5. BOUND ON NUMBER OF BOXES

In this section, we bound the number of boxes produced by our algorithm. All the proofs for this section are found in Appendix B.

```
Bisect(C)
  OUTPUT: a set of components containing all
      the non-adventitious roots in C
      (but possibly some adventitious ones)
  Initialize a Union-Find data structure U
          for boxes.
  For each constituent box B of C
     For each child B' of B
        If (T̃^G_0(Δ(B')) fails)
           U.add(B')
           For each box B'' ∈ U adjacent to B'
              U.union(B', B'')
  Initialize Q to be empty.
  specialFlag ← true
  If (U has only one connected component)
     specialFlag ← false
  For each connected component C' of U
     If (C' intersects B_0) // C' ≠ adventitious
        If (specialFlag)    N_{C'} = 4
        Else    N_{C'} = max{4, √N_C}
        Q.add(C')
     Else Q_dis.add(C')
  Return Q
```

Figure 3: Bisection Step

The goal is to bound the number of all the constituent boxes of the components in $\mathcal{T}_{comp}$. But, in anticipation of the following complexity analysis, we want to consider an **augmented component tree** $\widehat{\mathcal{T}}_{comp}$ instead of $\mathcal{T}_{comp}$.

Let $\widehat{\mathcal{T}}_{comp}$ be the extension of $\mathcal{T}_{comp}$ in which, for each confined adventitious components in $\mathcal{T}_{comp}$, we (conceptually) continue to run our algorithm until they finally produce output components, i.e., leaves of $\widehat{\mathcal{T}}_{comp}$. As before, these leaves have at most 9 constituent boxes.

Since $C' \to C$ denote the parent-child relation, a path in $\mathcal{T}_{comp}$ may be written

$$P = (C_1 \to C_2 \to \cdots \to C_s). \tag{6}$$

We write w_i, R_i, N_i, etc, instead of $w_{C_i}, R_{C_i}, N_{C_i}$, etc.

A component C is **special** if C is the root or a leaf of $\widehat{\mathcal{T}}_{comp}$, or if $\#(C) < \#(C')$ with C' the parent of C in $\widehat{\mathcal{T}}_{comp}$; otherwise it is **non-special**. This is a slight variant of [3].

We call P a **non-special path led by** C_1, if each C_i $(i = 2, \ldots, s)$ is non-special, i.e., $\#(C_i) = \#(C_{i-1})$. The **special component tree** $\mathcal{T}^*_{comp}$ is obtained from $\widehat{\mathcal{T}}_{comp}$ by eliminating any non-special components while preserving the descendent/ancestor relationship among special nodes.

Define $s_{\max}$ to be the maximum length of a non-special path in $\widehat{\mathcal{T}}_{comp}$.

LEMMA 7.

$$s_{\max} = O\Big(\log n + \log\log \frac{w(B_0)}{\varepsilon}\Big).$$

Charging function $\phi_0(B)$. For each component C, define the **root radius** of C to be $r_C := \mathrm{rad}(\mathcal{Z}(C))$, that is the radius of the smallest disc enclosing all the roots in C. We are ready to define a charging function ϕ_0 for each box B in the components of $\widehat{\mathcal{T}}_{comp}$: Let $C_B \in \widehat{\mathcal{T}}_{comp}$ be the component of which B is a constituent box. Let ξ_B be any root

ROOT CLUSTERING ALGORITHM
Input: Polynomial $F(z)$, box $B_0 \subseteq \mathbb{C}$ and $\varepsilon > 0$
Output: Components in Q_{out} representing natural ε-clusters of $F(z)$ in $2B_0$.

▷ *Initialization*
$Q_{out} \leftarrow Q_1 \leftarrow Q_{dis} \leftarrow \emptyset$.
$Q_0 \leftarrow \{(5/4)B_0\}$ // initial component
▷ *Preprocessing*
While Q_0 is non-empty
  $C \leftarrow Q_0.pop()$
  If (C is confined and $W_C \leq w(B_0)/2$)
    $Q_1.add(C)$
  Else $Q_0.add(Bisect(C))$
▷ *Main Loop*
While Q_1 is non-empty
  $C \leftarrow Q_1.pop()$ // C has the largest W_C in Q_1
  If ($4\Delta_C \cap C' = \emptyset$ for all $C' \in Q_1 \cup Q_{dis}$)
    $k_C \leftarrow \widetilde{T}^G_*(\Delta_C)$
    If ($k_C > 0$) // Note: $k_C \neq 0$.
      If ($W_C \geq \varepsilon$)
        $C' \leftarrow Newton(C)$
        If ($C' \neq \emptyset$)
          $Q_1.add(C')$; Continue
      Else if ($W_C \leq 3w_C$) // C is compact
        $Q_{out}.add(C)$; Continue
  $Q_1.add(Bisect(C))$
Return Q_{out}

Figure 4: Clustering Algorithm

in $2B$. There are two cases: (i) If C_B is a confined component, there is a unique maximum path in $\widehat{\mathcal{T}}_{comp}$ from C_B to a confined leaf E_B in $\widehat{\mathcal{T}}_{comp}$ containing ξ_B. Define $\phi_0(B)$ to be the first special component C along this path such that

$$r_C < 3w_B. \tag{7}$$

where w_B is the width of B. (ii) If C_B is not confined, it means that C is a component in the preprocessing stage. In this case, define $\phi_0(B)$ to be the largest natural ε-cluster containing ξ_B. Notice that $\phi_0(B)$ is a special component in (i) but a cluster in (ii).

LEMMA 8. *The map ϕ_0 is well-defined.*

Using this map, we can now bound the number of boxes.

LEMMA 9. *The total number of boxes in all the components in $\widehat{\mathcal{T}}_{comp}$ is*

$$O(t \cdot s_{\max}) = O(\#(2B_0) \cdot s_{\max})$$

with $t = |\{\phi_0(B) : B \text{ is any box in } \widehat{\mathcal{T}}_{comp}\}|$.

This improves the bound in [3] by a factor of $\log n$.

6. BIT COMPLEXITY

Our goal is to prove the bit-complexity theorem stated in the Introduction. All proofs are found in Appendix C.

The road map is as follows: we will charge the work of each box B (resp., component C) to some natural ε-cluster denoted $\phi(X)$ (resp., $\phi(C)$). We show that each cluster $\phi(X)$ (X is a box or a component) is charged $\widetilde{O}(1)$ times. Summing up over these clusters, we obtain our bound.

We may assume $\overline{\log}(B_0) = O(\tau_F)$ since Cauchy's root bound implies that any root z_i satisfies $|z_i| \leq 1 + 4 \cdot 2^\tau$, thus we can replace B_0 by $B_0 \cap B(0, 2 + 8 \cdot 2^\tau)$.

Cost of $\widetilde{T}^G$-tests and Charging function $\phi(X)$: Our algorithm performs 3 kinds of $\widetilde{T}^G$-tests:

$$\widetilde{T}^G_*(\Delta_C), \quad \widetilde{T}^G_{k_C}(\Delta'), \quad \widetilde{T}^G_0(\Delta(B)) \tag{8}$$

respectively appearing in the main loop, the Newton Step and the Bisection Step. We define the **cost** of processing component C to be the costs in doing the first 2 tests in (8), and the **cost** of processing a box B to be the cost of doing the last test. Note that the first 2 tests do not apply to the non-confined components (which appear in the preprocessing stage only), so there is no corresponding cost.

We next "charge" the above costs to natural ε-clusters. More precisely, if X is a confined component or any box produced in the algorithm, we will charge its cost to a natural ε-cluster denoted $\phi(X)$: (a) For a special component C, let $\phi(C)$ be the natural ε-cluster $\mathcal{Z}(C')$ where C' is the confined leaf of $\mathcal{T}^*_{comp}$ below C which minimizes the length of path from C to C' in $\mathcal{T}^*_{comp}$. (b) For a non-special component C, we define $\phi(C)$ to be equal to $\phi(C')$ where C' is the first special component below C. (c) For a box B, we had previously defined $\phi_0(B)$ (see Section 5). There are two possibilities: If $\phi_0(B)$ is defined as a special component, then $\phi(\phi_0(B))$ was already defined in (a) above, so we let $\phi(B) := \phi(\phi_0(B))$. Otherwise, $\phi_0(B)$ is defined as a natural ε-cluster, and we let $\phi(B) = \phi_0(B)$.

LEMMA 10. *The map ϕ is well-defined.*

Define $\widehat{S}$ to be the range of ϕ, so it is a set of natural ε-clusters. The clusters in $\widehat{S}$ are of two types: those defined by the confined leaves of $\widehat{\mathcal{T}}_{comp}$, and those largest ε-clusters of the form $\phi(B)$ with B in non-confined components.

LEMMA 11. *Each natural ε-cluster in $\widehat{S}$ is charged $O(s_{\max} \log n)$ times, i.e., $\widetilde{O}(1)$ times.*

We are almost ready to prove the theorems announced in Section 1.1. Theorem A is easier to prove if we assume that the initial box B_0 is **nice** in the following sense:

$$\max_{z \in 2B_0} \overline{\log}(z) = O(\min_{z \in 2B_0} \overline{\log}(z)). \tag{9}$$

Then the following lemma bounds the cost of processing X where X is a box or a component.

LEMMA 12. *If the initial box is nice, the cost of processing X (where X is a box or a component) is bounded by*

$$\widetilde{O}(n \cdot L_D)$$

bit operations with $D = \phi(X)$ and with L_D defined in (2). Moreover, an L_D-bit approximation of F is required.

Using this lemma, we could prove Theorem A of Section 1.1 under the assumption that B_0 is nice. The appendix will prove Theorem A holds even if B_0 is not nice; the proof of Theorems B is also found in the appendix. In [3], the complexity bound for global root isolation is reduced to the case where B_0 is centered at the origin. This requires a global pre-processing step. It is unclear that we can adapt that pre-processing to our local complexity analysis.

7. CONCLUSION

This paper initiates the investigation of the local complexity of root clustering. It modifies the basic analysis and techniques of [3] to achieve this. Moreover, it solves a problem left open in [3], which is to show that our complexity bounds can be achieved without adding a preprocessing step to search for "nice boxes" containing roots.

We mention some open problems. Our Theorem A expresses the complexity in terms of local geometric parameters; how tight is this? Another challenge is to extend our complexity analysis to analytic root clustering [31].

8. REFERENCES

[1] J. Abbott. Quadratic interval refinement for real roots. *Communications in Computer Algebra*, 28:3–12, 2014. Poster at the ISSAC 2006.

[2] R. Becker. The Bolzano Method to isolate the real roots of a bitstream polynomial. Bachelor Thesis, Uni. Saarland, Saarbruecken, Germany, May 2012.

[3] R. Becker, M. Sagraloff, V. Sharma, and C. Yap. A simple near-optimal subdivision algorithm for complex root isolation based on Pellet test and Newton iteration. *arXiv:1509.06231v3 [cs.NA]*, Sept. 2015. 51 Pages. Submitted to Journal.

[4] M. Burr and F. Krahmer. SqFreeEVAL: An (almost) optimal real-root isolation algorithm. *J. Symbolic Computation*, 47(2):153–166, 2012.

[5] M. Burr, F. Krahmer, and C. Yap. Continuous amortization: A non-probabilistic adaptive analysis technique. ECCC, TR09(136), Dec 2009.

[6] M. A. Burr. Continuous amortization and extensions: With applications to bisection-based root isolation. *J. Symbolic Computation*, 77:78–126, 2016.

[7] G. E. Collins and A. G. Akritas. Polynomial real root isolation using Descartes' rule of signs. In R. D. Jenks, editor, *Proc. 1976 ACM Symp. on Symbolic and Algebraic Comp.*, pp. 272–275. ACM Press, 1976.

[8] J. H. Davenport. Computer algebra for cylindrical algebraic decomposition. Tech. Rep., The Royal Inst. of Technology, Dept. of Numerical Analysis and Comput. Sci., S-100 44, Stockholm, Sweden, 1985.

[9] Z. Du, V. Sharma, and C. Yap. Amortized bounds for root isolation via Sturm sequences. In D. Wang and L. Zhi, eds., *Symbolic-Numeric Computation*, Trends in Mathematics, pp. 113–130. Birkhäuser, 2007.

[10] A. Eigenwillig, V. Sharma, and C. Yap. Almost tight complexity bounds for the Descartes method. In *31st ISSAC*, pp. 71–78.

[11] I. Z. Emiris, V. Y. Pan, and E. P. Tsigaridas. Algebraic algorithms. In *Computing Handbook, 3rd Edition: Computer Science and Software Engineering*, pages 10: 1–30. Chapman and Hall/CRC, 2014.

[12] N. Kamath. Subdivision algorithms for complex root isolation: Empirical comparisons. MSc thesis, Oxford University, Oxford Computing Lab, Aug. 2010.

[13] N. Kamath, I. Voiculescu, and C. Yap. Empirical study of an evaluation-based subdivision algorithm for complex root isolation. In *4th SNC Workshop*, pp. 155–164, 2011.

[14] P. Kirrinnis. Polynomial factorization and partial fraction decomposition by simultaneous Newton's iteration. *J. of Complexity*, 14:378–444, 1998.

[15] J. M. Lane and R. F. Riesenfeld. Bounds on a polynomial. *BIT*, 21:112–117, 1981.

[16] M. Marden. *The Geometry of Zeros of a Polynomial in a Complex Variable*. Amer. Math. Soc., 1949.

[17] J. McNamee and V. Pan. *Numerical Methods for Roots of Polynomials, Part 2*. Elsevier, Amsterdam, 2013.

[18] K. Mehlhorn, M. Sagraloff, and P. Wang. From approximate factorization to root isolation with application to cylindrical algebraic decomposition. *J. Symbolic Computation*, 66:34–69, 2015.

[19] C. A. Neff and J. H. Reif. An efficient algorithm for the complex roots problem. *J. of Complexity*, 12:81–115, 1996.

[20] V. Y. Pan. Solving a polynomial equation: some history and recent progress. *SIAM Review*, 39(2):187–220, 1997.

[21] V. Y. Pan. Approximating polynomial zeros: Modified quadtree (Weyl's) construction and improved Newton's iteration. *J.Complexity*, 16(1):213–264, 2000.

[22] V. Y. Pan. Univariate polynomials: Nearly optimal algorithms for numerical factorization and root-finding. *J. Symb. Comput.*, 33(5):701–733, 2002.

[23] J. Renegar. On the worst-case arithmetic complexity of approximating zeros of polynomials. *Journal of Complexity*, 3:90–113, 1987.

[24] F. Rouillier and P. Zimmermann. Efficient isolation of [a] polynomial's real roots. *J. Computational and Applied Mathematics*, 162:33–50, 2004.

[25] M. Sagraloff. When Newton meets Descartes: A simple and fast algorithm to isolate the real roots of a polynomial. In *37th ISSAC*, pages 297–304, 2012.

[26] M. Sagraloff and K. Mehlhorn. Computing real roots of real polynomials. *J. Symbolic Computation*, 2015.

[27] M. Sagraloff and C. K. Yap. A simple but exact and efficient algorithm for complex root isolation. In *36th ISSAC*, pages 353–360, 2011.

[28] A. Schönhage. The fundamental theorem of algebra in terms of computational complexity, 1982. Manuscript, Department of Mathematics, University of Tübingen. 2004 Update with typo corrections and an appendix.

[29] V. Sharma and C. Yap. Near optimal tree size bounds on a simple real root isolation algorithm. In *37th ISSAC*, pp. 319 – 326, 2012.

[30] J.-C. Yakoubsohn. Finding a cluster of zeros of univariate polynomials. *Journal of Complexity*, 16(3):603–638, 2000.

[31] C. Yap, M. Sagraloff, and V. Sharma. Analytic root clustering: A complete algorithm using soft zero tests. In *Computability in Europe (CiE2013)*, LNCS vol. 7921, pages 434–444, Heidelberg, 2013. Springer.

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

[33] C. K. Yap. In praise of numerical computation. In *Efficient Algorithms*, volume 5760 of *Lect. Notes in C.S.*, pp. 308–407. Springer-Verlag, 2009.

APPENDIX

We have omitted the three appendices which may be found in our full paper: Appendix A contain proofs for Section 2. Similarly, Appendix B and C are for Sections 5 and 6.