Computing real roots of real polynomials . . . . . . and now for real!

Alexander Kobel
Max-Planck-Institut
für Informatik
Campus E1 4
66123 Saarbrücken, Germany
alexander.kobel@mpi-inf.mpg.de

Fabrice Rouillier
INRIA Paris
Institut de Mathématiques de
Jussieu Paris Rive Gauche
CNRS UMR 7586
Université P. & M. Curie
4 place Jussieu 75252 Paris
Cedex 05, France
Fabrice.Rouillier@inria.fr

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

ABSTRACT

Very recent work introduces an asymptotically fast subdivision algorithm, denoted ANewDsc, for isolating the real roots of a univariate real polynomial. The method combines Descartes' Rule of Signs to test intervals for the existence of roots, Newton iteration to speed up convergence against clusters of roots, and approximate computation to decrease the required precision. ANewDsc achieves record bounds on the worst-case complexity for the considered problem, matching the complexity of Pan's method for computing all complex roots and improving upon the complexity of other subdivision methods by several magnitudes.

In the paper at hand, we report on an implementation of ANewDsc on top of the RS root isolator. At the current stage, RS is the most efficient realization of the classical Descartes method, and also constitutes the default real root solver for polynomials in Maple. We describe several crucial design changes within ANewDsc as well as within RS that have led to a high-performance implementation without harming the theoretical complexity of the underlying algorithm. Testing our implementation on numerous benchmark instances shows that the theoretical gain in performance of ANewDsc over other subdivision methods also transfers into practice. Experiments show that our new implementation outperforms RS by magnitudes for notoriously hard instances with clustered roots. For all other instances, there is almost no overhead due to the integration of additional techniques.

CCS Concepts

• **General and reference → Design;** *Experimentation; Performance* • **Mathematics of computing → Solvers; Computations on polynomials; Interval arithmetic;** *Mathematical software performance; Arbitrary-precision arithmetic*

ISSAC 2016 July 20-22, 2016, Waterloo, Ontario, Canada.
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN ???-?-????-????-?. . . $??.??
DOI: 10.1145/???????.???????

Keywords

real roots; univariate polynomials; root finding; root isolation; Newton's method; Descartes method; approximate arithmetic; certified computation

1. INTRODUCTION

Computing the real roots of a univariate polynomial with real coefficients is one of the fundamental tasks in numerics and computer algebra, and numerous methods have been proposed to solve this problem. The leading general-purpose solvers in practice are based on subdivision algorithms that rely on Descartes' Rule of Signs to test for the existence of roots in a certain interval.

The computation for an input polynomial $P \in \mathbb{Q}[x]$ can be considered as a binary tree where each node corresponds to an interval $I = (a, b)$ and a polynomial $P_I = (x+1)^n P(\frac{ax+b}{x+1})$. The number v_I of sign changes in the coefficient sequence of P_I then exceeds the number of roots contained in I by an even non-negative number. In the original algorithm [2], the two children of a node are obtained by a simple bisection on the interval and relative transformations on the polynomials, which yields the polynomial P_I. A node is a leaf if $v_I = 0$ or $v_I = 1$; in the first case, I contains no root, whereas, in the latter case, the interval is isolating for a real root. Considerable efforts have been taken to improve the worst-case complexity of different variants of Descartes methods, and to provide efficient implementations: different traversal orders of the subdivision tree to optimize the memory usage [10], the use of interval arithmetic [9], new strategies to optimize the main transformations [17], extensions to polynomials with approximate coefficients [11, 6], and many others.

Nevertheless, most formulations still suffer from two main deficiencies: the subdivision strategies only achieve *linear* convergence against the roots. In presence of clusters, those approaches require a large number of subdivisions to separate the roots. Furthermore, an unguarded choice of the subdivision points can impose an unnecessarily large precision demand. Even worse, completeness is not guaranteed for polynomials whose coefficients can only be approximated.

Both shortcomings have been resolved in [20]: The proposed algorithm ANewDsc combines the bisection strategy with Newton iteration to achieve *quadratic* convergence against clusters, thus compressing long chains of intervals I with a constant v_I to only logarithmic length (compared to the length when considering bisection only). In addition, intervals are split at *admissible points,* where the polynomial takes a (relatively) large absolute value. This allows to keep the precision demand of the computation to a minimum, improving by an order of magnitude over previous approaches. Due to the exclusive use of approximate arithmetic, the method also generalizes to (square-free) polynomials with arbitrary real coefficients. Both the number of subdivision steps and the precision demand is near-optimal, and the bound on its bit complexity matches the record bound [12] that is implied by an algorithm based on Pan's near-optimal method [14] for approximate polynomial factorization. We will provide more specific bounds in Section 2.

Unfortunately, it often happens that asymptotically fast algorithms are extremely hard to implement or do not exhibit their theoretical performance in practice. The contribution of this paper is to show that, for the problem of real root computation, thus closing the so far existing gap between theory and practice: We report on an implementation of ANewDsc on top of the Descartes-based real root finder inside the RS library. It preserves the key features of the implementation in RS, which are a close-to-optimal memory consumption and the intensive use of adaptive multiprecision interval arithmetic [16],

We present our design changes both to RS and within ANewDsc that make it possible to achieve significant performance gains on hard instances on the one hand and proven complexity guarantees on the other hand without sacrificing efficiency for small or intrinsically easy instances. The analysis is supported by benchmark results which show that ANewDsc can defy the leading general-purpose solvers for real roots in their special domains, and outperforms the existing implementations on notoriously hard instances.

2. THE DESCARTES METHOD AND THE ANewDsc ALGORITHM

We briefly review the classical Descartes method [2] as well as the algorithm ANewDsc as introduced in [20]. Given an interval $\mathcal{I} = (a_0, b_0) \subset \mathbb{R}$ and a polynomial

$$P(x) = \sum\nolimits_{i=0}^{n} p_i \cdot x^i \in \mathbb{R}[x], \qquad (1)$$

the Descartes method recursively subdivides $\mathcal{I}$ into equally sized intervals until each subinterval $I = (a, b) \subset \mathcal{I}$ has either been shown to contain no root or exactly one root of P. In order to test an interval for the existence of a root of P, a coordinate transformation $\phi_I : x \mapsto \frac{ax+b}{x+1}$ is considered, which maps $\mathbb{R}^+$ one-to-one onto the interval I. Then, Descartes' Rule of Signs applied to the polynomial

$$P_I(x) := \sum\nolimits_{i=0}^{n} p_{I,i} x^i := (x+1)^n \cdot P\left(\frac{ax+b}{x+1}\right) \qquad (2)$$

states that the number of sign changes $\operatorname{var}(P, I) := \operatorname{var}(P_I) := \operatorname{var}(p_{I,0}, \ldots, p_{I,n})$ in the coefficient sequence of P_I exceeds the number m_I of roots (counted with multiplicity) of P in I by a non-negative even integer. In other words, $m_I \leq \operatorname{var}(P, I)$ and $m_I \equiv \operatorname{var}(P, I) \mod 2$. Another important property is that the function $\operatorname{var}(\cdot)$ is variation diminishing [5]. That is, for any disjoint subintervals I_1 and I_2 of I, it holds that $\operatorname{var}(P, I_1) + \operatorname{var}(P, I_2) \leq \operatorname{var}(P, I)$. In each step of the recursion within the Descartes method, the number $\operatorname{var}(P, I)$ is computed. If $\operatorname{var}(P, I) = 0$, the interval I is discarded. If $\operatorname{var}(P, I) = 1$, I is stored as isolating. Intervals that yield more than one sign variation (i.e., $\operatorname{var}(P, I) > 1$) are further subdivided into two equally sized sub-intervals. In addition, it is checked whether the subdivision point, that is the midpoint $m(I) = \frac{a+b}{2}$ of I, is a root of P.

Algorithm 1: Classical Descartes Method

```
INPUT: A polynomial $P(x)$ as in (1) and an interval $\mathcal{I}$.
OUTPUT: Disjoint isolating intervals $I_1, \ldots, I_{m_\mathcal{I}}$
        for all real roots of $P$ in $\mathcal{I}$.
• Initialize the list of active intervals to $\mathcal{A} := \{\mathcal{I}\}$.
• While $\mathcal{A} \neq \emptyset$:
    ▷ Remove an arbitrary $I = (a, b)$ from $\mathcal{A}$.
    ▷ If $\operatorname{var}(P, I) = 0$, discard $I$ and continue.
    ▷ If $\operatorname{var}(P, I) = 1$, add $I$ to $\mathcal{O}$ and continue.
    ▷ If $P(m(I)) = 0$, add $\{m(I)\}$ to $\mathcal{O}$.
    ▷ Add $(a, m(I))$ and $(m(I), b)$ to $\mathcal{A}$.
• Return $\mathcal{O}$.
```

If the polynomial P contains only simple roots in $\mathcal{I}$, the Descartes method yields isolating intervals for all these roots; otherwise, it converges towards the roots, but does not terminate. If $\mathcal{I}$ is chosen large enough to contain all real roots, and all these roots are simple, the algorithm isolates all real roots of P. The proof for termination rests on the well known One- and Two-Circle Theorems [5, 13], which imply that the width $w(I)$ of any produced interval I is lower bounded by $\frac{1}{2} \cdot \min_{i=1,\ldots,n} \max(\operatorname{sep}(z_i, P), \operatorname{dist}(z_i, I))$, which is larger than or euqal to $\operatorname{sep}(P)/2$. Here, $z_1, \ldots, z_n$ denote the complex roots of P, $\operatorname{sep}(z_i, P) := \min_{j \neq i} |z_i - z_j|$ is defined as the *separation of* z_i, $\operatorname{sep}(P) := \min_i \operatorname{sep}(z_i, P)$ as the *separation of* P, and $\operatorname{dist}(z_i, I)$ as the distance from z_i to I.

Regarding the worst-case complexity of the above algorithm, we focus, only for simplicity, on the so-called *benchmark problem* of computing all real roots of a square-free polynomial P of degree n with *integer* coefficients of absolute value less than 2^τ. Nevertheless, we aim to stress the fact that, for polynomials with arbitrary real coefficients, more general bounds are known [19, 20], which are expressed in terms of the separations and the absolute values of the roots of P, thus being more adaptive and meaningful. We denote by $\mathcal{T}$ the subdivision tree whose nodes are the intervals I produced by the algorithm. Then, from the sign variation diminishing property of $\operatorname{var}(\cdot)$ and the lower bound on the width of the produced intervals, it follows that $\mathcal{T}$ has width at most $2n$ and height $O(\tau + \log \operatorname{sep}(P)^{-1})$. Since $\log \operatorname{sep}(P)^{-1} = \tilde{O}(n\tau)$, this yields the bound $\tilde{O}(n^2\tau)$ on the total size of $\mathcal{T}$. However, a more refined argument [7, 5], which takes into account the fact that $\sum_{i=1}^{n} \log \operatorname{sep}(z_i, P)^{-1} = \tilde{O}(n\tau)$, even shows that $|\mathcal{T}| = \tilde{O}(n\tau)$. The coefficients of the polynomials P_I have bitsize bounded by $\tilde{O}(n^2\tau)$, hence the precision demand for exactly computing P_I is also bounded by $\tilde{O}(n^2\tau)$. This yields the bound $\tilde{O}(n^3\tau)$ for the bit complexity of computing P_I, even when using asymptotically fast methods for multiplication and Taylor shift computation. We conclude that the cost of the Descartes method is bounded by $\tilde{O}(n^4\tau^2)$ bit operations, which matches the bound as achieved by many other popular subdivision algorithms for real root computation such as the continued fraction method [1, 21], the Bolzano method [22], or the Sturm method [3]. We further remark that the bound $\tilde{O}(n^4\tau^2)$ is not just an upper bound but actually a tight bound for the latter mentioned algorithms; see [7, 1, 4].

The main strengths of the Descartes method are its simplicity and the fact that, for most instances, its running time is much better than reflected by the worst case bounds. However, the method has two major shortcomings. First, in order to determine the sign of the coefficients of P_I as well as the sign of P at the midpoint of I, exact arithmetic is needed in general. If the given input is defined over the rational numbers, the computations can be carried out exactly. However, this may lead to an unnecessarily large precision demand (i.e. $\tilde{O}(n^2\tau)$ for the benchmark problem), which considerably exceeds the actual precision demand (which is $\tilde{O}(n\tau)$ in the worst-case) that is needed to compute isolating interval; see [19, 20] for more details. Even worse, if the coefficients of P can only be approximated, then computing the sign of the coefficients of P_I is not feasible, at least in general. Another shortcoming of the Descartes method is that it only achieves linear convergence against the roots. For most polynomials, this is not critical as the separations of the roots are large, thus resulting in a subdivision tree of small height. However, if roots appear in clusters, the algorithm has to perform numerous subdivision steps in order to separate distinct roots from each other. In fact, there exist polynomials[1] for which the Descartes method produces a sequence of intervals I_j of length $\Omega(n\tau)$ such that $\operatorname{var}(P, I_j)$ stays invariant.

In [20], an algorithm denoted[2] ANewDsc has been introduced that addresses both of the above mentioned shortcomings. Here, we only sketch the main ideas to an extent as needed in the discussion of its implementation. At some points, we also simplified the presentation at the cost of mathematical rigorousness by skipping some rather technical details. In contrast, our implementation takes into account all details, thus being certified and complete. ANewDsc is similar to the classical Descartes method in the way that it recursively subdivides a given interval $\mathcal{I}$ and that is uses Descartes' Rule of Signs to test for roots. However, instead of choosing a fixed subdivision point m (typically the midpoint $m(I)$ of an interval I) in each step, it chooses a nearby, so-called *admissible point* m^*, where $|P|$ becomes large. More precisely, for a point m contained in some interval $I = (a, b)$, a multiple $N := c \cdot n$ of n with $c \in \mathbb{Z}_{\geq 1}$, and some positive value ϵ, we call a point

$$m^* \in m[\epsilon; N] := \{m_i := m + \tfrac{i}{c} \cdot \epsilon \text{ for } i = -\lceil \tfrac{N}{2} \rceil, \ldots, \lceil \tfrac{N}{2} \rceil\}$$

admissible with respect to the multipoint $m[\epsilon; N]$ *(or just admissible)* if $|P(m^*)| \geq \frac{1}{4} \cdot \max_i |P(m_i)|$. In [20, Section 2.2], we choose $N = n$, however, the algorithm as well as its analysis works with any constant multiple of N.

Notice that the above algorithm terminates and that $P(m^*) \neq 0$ for the returned admissible point m^*. Indeed, this follows from that fact $m[\epsilon; N]$ contains at least $n + 1$ points, and thus at least one of the points m_i must have distance $\epsilon/2c$ or more to each of the roots of P. Now suppose that ϵ is chosen small enough, that is, we have $\epsilon \approx w(I)/4$ in the linear (bisection) step of ANewDsc and $\epsilon \approx w(I)/4N_I$ for all admissible point-computations in the Newton-Test. Then, according to the analysis from [20], the subdivision

Algorithm 2: Find Admissible Point

```
INPUT: A polynomial $P(x)$ as in (1) and
       a multipoint $m[\epsilon; N]$.
OUTPUT: An admissible point $m^* \in m[\epsilon; N]$.
• For $\rho = 2, 4, 8, \ldots$ :
    ▷ For all $m_i \in m[\epsilon; N]$, compute approximations $\tilde{v}_i$
      of $P(m_i)$ with $|\tilde{v}_i - P(m_i)| < 2^{-\rho}$.
    ▷ Determine a point $m_{i_0} = m_i$ that maximizes $|\tilde{v}_i|$.
    ▷ If $|\tilde{v}_{i_0}| > 2^{-\rho+1}$, return $m^* := m_{i_0}$.
```

tree induced by ANewDsc does not change in considerable manner (in the worst case) when passing from a canonical point m to an admissible point $m^* \in m[\epsilon; N]$ in each iteration. Hence, for simplicity, we will not further specify ϵ (nor N) throughout the following considerations and just assume that ϵ is chosen small enough (as in [20]). In this context, we just say that a point m^* is *admissible for* m.

By choosing admissible subdivision points, we can guarantee that the distance between the endpoints of any interval $I = (a, b)$ produced by ANewDsc and the roots of P is not too small. As a consequence, it is now possible to test I for the existence of roots using approximate arithmetic only. Indeed, there exists some test, which we call the 01-Test[3] which runs with a precision bounded by $\tilde{O}(\log(n + n|a| + n|b| + |f(a)|^{-1} + |f(b)|^{-1} + \|P\|_\infty))$, and which fulfills the following properties:

(1) If $\operatorname{var}(P, I) = 0$, the 01-Test returns the value 0, which indicates that I contains no root.
(2) If $\operatorname{var}(P, I) = 1$, the 01-Test returns the value 1, which indicates that I is isolating for a simple real root of P.
(3) If $\operatorname{var}(P, I) > 1$, the 01-Test might return any value from $\{-1, 0, 1\}$.

In [20], it has been shown that, for the benchmark problem, the precision demand never exceeds the bound $\tilde{O}(n\tau)$, which is by a factor n better than the precision demand needed by the classical Descartes method.

In order to overcome the second shortcoming of the Descartes method, ANewDsc combines bisection with Newton iteration. For this, it uses a trial and error approach, called "Newton-Test", to speed up convergence towards clusters of roots. In each iteration, the Newton-Test aims to replace an interval $I = (a, b)$ by some sub-interval $I' = (a', b') \subset I$ of width $w(I') \approx w(I)/N_I$ such that I' contains all roots of P in I. Here, N_I denotes a parameter that corresponds to the actual speed of convergence. Initially, N_I is set to 4. If the Newton step succeeds, we define $N_{I'} := N_I^2$. In case of failure, we fall back to bisection, that is, I is subdivided into two (almost) equally-sized subintervals I_ℓ and I_r, and we define $N_{I_\ell} = N_{I_r} := \max(4, \sqrt{N_I})$. In [20, Section 3.2], it has been shown that the Newton step succeeds under guarantee if there exists a sufficiently small cluster $\mathcal{C}$ of roots of P that is centered at some point in I and that is is sufficiently well-separated from the remaining roots of P.

The method runs in three steps. In the first step, it computes the multiplicity k of such a cluster.[4] Then, in the second step, it performs a corresponding Newton iteration

[1] As an example, consider the Mignotte-like polynomials from Table 1, which have a cluster consisting of two real roots with pairwise distance $2^{-\Omega(n\tau)}$.

[2] The algorithm is an approximate arithmetic variant of the algorithm NewDsc from [18], which exclusively uses exact arithmetic. The acronym ANewDsc should be read as "A New Descartes" or, alternatively, as "Approximate Arithmetic Newton-Descartes."

[3] The 01-Test is split into two separate subroutines. Both are modified variants of the classical sign variation tests, where it is checked whether $\operatorname{var}(P, I) = 0$ or $\operatorname{var}(P, I) = 1$.

[4] In fact, it only computes some rational value $\tilde{k}$. Under the assumption that a cluster $\mathcal{C}$ as above exists, we have $k \approx \tilde{k}$ and,

to obtain a λ with $|\lambda - z_i| < w(I)/2N_I$ for each root $z_i \in \mathcal{C}$. Finally, the method aims to validate the approximation λ of the cluster $\mathcal{C}$. For this, we apply the 01-Test to the intervals (a, a') and (b, b'), where $I' = (a', b') \subset I$ is an interval of width $w(I') \approx w(I)/N_I$ centered at λ. If the latter test yields the value 0 for both intervals (a, a') and (b, b'), it follows that I' contains all roots that are contained in I, in which case we say that the Newton-Test succeeds. Notice that, even in case of success, we may not conclude that there exists a cluster of k roots, however, we may conclude that I may be replaced by I' without discarding any root.

Algorithm 3: Newton-Test

```
INPUT: A polynomial $P(x)$ as in (1), an interval $I = (a, b)$,
       and an $N_I \in \mathbb{N}$ of the form $N_I = 2^{2^l}$ with $l \in \mathbb{N}_{\geq 1}$.
OUTPUT: FALSE or an interval $I' \subset I$, with $\frac{N_I w(I')}{w(I)} \in [\frac{1}{8}, 1]$,
        that contains all roots of $P$ in $I$.
• For $j \in \{1, 2, 3\}$, let $\xi_j := a + \frac{j}{4} \cdot w(I)$,
  compute admissible points $\xi_j^*$ for $\xi_j$ and
  the Newton correction terms $v_j := P(\xi_j^*)/P'(\xi_j^*)$.
• For all pairs $i, j \in \{1, 2, 3\}$ with $i < j$:
    ▷ Compute approximations $\tilde{\lambda}_{i,j}$ of
        $\lambda_{i,j} := \xi_i^* + \tilde{k} \cdot v_i$, where $\tilde{k} := (\xi_j^* - \xi_i^*)/(v_i - v_j)$,
      with $|\tilde{\lambda}_{i,j} - \lambda_{i,j}| \leq 1/32N_I$.
    ▷ If $\tilde{\lambda}_{i,j} \notin [a, b]$, discard $(i, j)$. Otherwise, define
        $\ell_{i,j} := \lfloor (\tilde{\lambda}_{i,j} - a) \cdot 4N_I/w(I) \rfloor \in \{0, \ldots, 4N_I\}$,
        $a_{i,j} := a + \max\{0, \ell_{i,j} - 1\} \cdot w(I)/4N_I$, and
        $b_{i,j} := a + \min\{4N_I, \ell_{i,j} + 2\} \cdot w(I)/4N_I$.
      If $a_{i,j} = a$, set $a_{i,j}^* := a$, and if $b_{i,j} = b$, set $b_{i,j}^* := b$.
      Otherwise, compute admissible points $a_{i,j}^*$ and $b_{i,j}^*$
      for $a_{i,j}$ and $b_{i,j}$.
    ▷ If the 01-Test returns 0 for both intervals $(a, a_{i,j}^*)$
      and $(b_{i,j}^*, b)$, return $I' = (a_{i,j}^*, b_{i,j}^*)$.
      Otherwise, discard the pair $(i, j)$.
• Boundary test: If all pairs have been discarded,
  compute admissible points $m_\ell^*$ and $m_r^*$ for
  $m_\ell := a + w(I)/2N_I$ and $m_r := b - w(I)/2N_I$.
    ▷ If the 01-Test yields 0 for $(a, m_\ell^*)$, return $(a, m_\ell^*)$.
    ▷ If the 01-Test yields 0 for $(m_r^*, b)$, return $(m_r^*, b)$.
    ▷ Otherwise, return FALSE.
```

This is a simplified version of Newton- and Boundary-Test from [20]. For the sake of a concise presentation, some technical details (e.g. concerning the precision management) are omitted.

It should further be mentioned that the Newton-Test also integrates a so-called Boundary-Test, which, by default, always checks whether one of the two intervals $(a, a + w(I)/N_I)$ or $(b - w(I)/N_I, b)$ contains all roots that are contained in I. Again, for this, the 01-Test is applied to $(a + w(I)/N_I, b)$ and $(a, b - w(I)/N_I)$, respectively; see Algorithm 4 and [20, Section 3.2] for more details.

ANewDsc can be used to isolate the real roots of any square-free[5] polynomial $P \in \mathbb{R}[x]$ with arbitrary real coefficients. Indeed, due to the exclusive use of approximate arithmetic, only (sufficiently good) approximations of the input polynomial are needed. In [20], the complexity (i.e. the overall bit complexity as well as bounds on the number of

in particular, k equals the integer that is closest to $\tilde{k}$; see the definition in the pseudo-code for the Newton-Test.

[5]It is only necessary that all roots contained in the initial interval $\mathcal{I}$ are simple.

Algorithm 4: ANewDsc

```
INPUT: A polynomial $P(x)$ as in (1) and an interval $\mathcal{I}$.
OUTPUT: Disjoint isolating intervals $I_1, \ldots, I_{m_\mathcal{I}}$
        for all real roots of $P$ in $\mathcal{I}$.
• Initialize the list of active intervals to $\mathcal{A} := \{(\mathcal{I}, 4)\}$.
• While $\mathcal{A} \neq \emptyset$:
    ▷ Remove an arbitrary $(I, N_I)$ with $I = (a, b)$ from $\mathcal{A}$.
    ▷ If the 01-Test on $I$ returns 0,
      discard $I$ and continue.
    ▷ If the 01-Test on $I$ returns 1,
      add $I$ to $\mathcal{O}$ and continue.
    ▷ Quadratic (or Newton) step:
      If the Newton-Test on $P$ and $(I, N_I)$ returns an
      interval $I'$, add $(I', N_I^2)$ to $\mathcal{A}$ and continue.
    ▷ Linear (or bisection) step:
      Compute an admissible point $m^*$ for $m(I)$ and add
      $(I', N_{I'})$ and $(I'', N_{I''})$ to $\mathcal{A}$, where $I' = (a, m^*)$,
      $I'' = (m^*, b)$, and $N_{I'} := N_{I''} := \max(4, \sqrt{N_I})$.
• Return $\mathcal{O}$.
```

iterations and the precision demand) of ANewDsc is stated in terms of the degree of P and values that exclusively depend on the roots z_i of the polynomial, such as the product of the absolute values of all roots that are not contained in the unit disk or the product of the pairwise distances between any two roots. Again, restricting to the benchmark problem, it holds that the length of a sequence of intervals I_j with ivariant $\text{var}(P, I_j)$ is now upper bounded by $\tilde{O}(\log(n\tau))$ compared to $\tilde{O}(n\tau)$ for the classical Descartes method. This is due to the fact that the Newton-Test succeeds for all but $\tilde{O}(\log(n\tau))$ intervals I_j and that quadratic convergence is achieved in all other iterations. As a consequence, the overall size of the subdivision tree is bounded by $O(\text{var}(P, I) \cdot \log(n\tau)) = O(n \log(n\tau))$, which is near-optimal; see also [20, Theorems 27–29]. The bit-complexity of the algorithm is bounded by $\tilde{O}(n^3 + n^2\tau)$, which is by three magnitudes better than the bound for the Descartes method and comparable to the best bound known [12, 8] for the benchmark problem as achieved by an algorithm based on Pan's near-optimal method [15] for approximate polynomial factorization. The bound is obtained by an amortized analysis of the cost at each node in the subdivision tree. In the worst-case, the cost at a node is of size $\tilde{O}(n^2\tau)$, whereas, in average, it is of size $\tilde{O}(n(n + \tau))$. The precision demand is upper bounded by $\tilde{O}(n\tau)$ in the worst case and of size $\tilde{O}(n + \tau)$ in average.

3. IMPLEMENTING ANewDsc INSIDE RS

The RS library contains a generic framework [17] that allows to instantiate several variants of the Descartes method. The choices include several bisection variants as well as continued fraction-based subdivision. The default configuration, which we will denote as "RS" throughout the following considerations, achieves its high efficiency by a careful low-level implementation of the bottleneck subroutines as well as a few sophisticated and—at that time—innovative design choices.

RS uses a hybrid arithmetic strategy. Starting with a low precision (of 63 bits), all predicates are evaluated in interval arithmetic using MPFI [16]. Whenever a predicate cannot be conclusively decided within the working precision, the computation is interrupted and later resumed with a higher precision. When it is conceivable that exact arithmetic is less costly than the overhead for interval arithmetic at high precision, a heuristic can trigger exact computation; this facilitates further optimizations like (virtual) deflation of exact dyadic roots.

The realization of the 01-Test in RS is adapted to this interval setting: While we work with exact polynomials $f = \sum f_i x^i$ in theory, the computations are executed on coefficient-wise interval approximations $[f] = \sum [f]_i x^i$ of f such that $f_i \in [f]_i$ for all i. In particular, the value of $\operatorname{var}(f)$ is replaced by a (super-)set $\operatorname{var}([f]) = \{v^-, \dots, v^+\}$ of possible sign variations for the family of polynomials in $[f]$, that is, $\operatorname{var}(g) \in \operatorname{var}([f])$ for all $g \in [f]$. In this model, a sign variation test for a polynomial $[f]$ succeeds if and only if $\operatorname{var}([f]) = \{0\}$, $\operatorname{var}([f]) = \{1\}$, or $0, 1 \notin \operatorname{var}([f])$. Otherwise, the accuracy is not sufficient to decide the branching strategy for the current interval, and the precision is increased. According to the definition of the 01-Test and our strategy of choosing admissible points as subdivision points, it is guaranteed that the sign variation test for a polynomial $[f]$ eventually succeeds with a precision as stated in Section 2. In the theoretical description of ANewDsc, the 01-Test needs to execute the computation of $\operatorname{var}(\cdot)$ on two subintervals of I in order to guarantee convergence within the stated precision; see [20, Section 2.4] for details. However, a single call on I is sufficient to decide how to branch in most situations; we use this simplified test as a filter and only fall back to the exhaustive variant in the exceptional cases.

Another crucial difference between RS and the previous variants of the Descartes method is the traversal of the subdivision tree. In Collins' and Akritas' [2] formulation, the subdivision tree is explored in a depth-first search strategy. This traversal allows to obtain some of the intermediate polynomials with comparatively little computational effort from previous results, but comes at the expense of storing a potentially very long list of active nodes. On the other hand, Krandick's variant [9] with breadth-first traversal is more memory-efficient, but requires more expensive arithmetic operations; see [17, Section 3]. The trade-off which proved optimal for RS was a depth-first search traversal in a near-constant-memory variant, oblivious of intermediate results. To keep the arithmetic cost of the algorithm close to the optimum for bisection methods, RS uses specifically tailored subroutines for reconstructing the discarded intermediate polynomials. They are supplemented by a custom garbage collector that ensures that only an insignificant amount of time is spent for memory management. For details of those design choices, we refer to the original description of the RS framework [17, Section 4].

RS is a promising basis for an implementation of ANewDsc due to its high performance as a general-purpose solver and the availability in Maple. Also, the extensive use interval arithmetic offers the flexibility to design a numerical root solver for polynomials with real bitstream coefficients. However, to combine the theoretical improvements of the new algorithm with the long-evolved insights for a practical realization, modifications are required on both ends. In this section, we will describe the most crucial design decisions in this light.

3.1 Changes over the theoretical ANewDsc

Random sampling of pseudo-admissible points.

A tight theoretical analysis of the worst-case bit complexity of any variant of the Descartes method heavily relies on asymptotically fast algorithms for polynomial arithmetic. In practice however, many asymptotically fast methods only become effective for inputs larger than a threshold that is well beyond what can realistically be handled with state-of-the-art solvers. Until now, careful implementations of the naive algorithms are more efficient except for artificial counterexamples.

The most prominent places where such tools are used in ANewDsc are the basis transformations from (1) to (2) (so-called *Taylor shifts*) and the approximate multipoint evaluations within the admissible point selections. With the advanced methods, both subroutines require only a near-linear number of arithmetic operations in the coefficient ring, compared to quadratically many for naive approaches. The Taylor shifts are intrinsic to any Descartes method (although we will describe a partial remedy at the end of Section 3.2). But the multipoint evaluations in this context can be avoided: In contrast to choosing a point m^* with guaranteed value $|P(m^*)| \geq \frac{1}{4} \cdot \max_i |P(m_i)|$ among a multipoint $m[\epsilon; N]$, we randomly sample in $m[\epsilon; N]$ until we find a so-called *pseudo-admissible point* where P is only required to take a non-zero value. The evaluation is done with fixed precision which is doubled for every sample.

Algorithm 5: Find Pseudo-Admissible Point

```
INPUT: A polynomial $P(x)$ as in (1) and
       a multipoint $m[\epsilon; N]$ with $N := 4n$.
OUTPUT: A point $m' \in m[\epsilon; N]$ with value $P(m') \neq 0$.
• For $\rho = 63, 127, 255, \dots$ :
    ▷ Pick a random $m'$ among the points in $m[\epsilon; N]$.
    ▷ Evaluate $[v'] = P(m')$ with interval arithmetic
      in precision $\rho$.
    ▷ If $0 \notin [v']$, return $m'$.
```

In the implementation, we give a slight preference to the canonical subdivision point m if this point can be shown to be pseudo-admissible within low precision. The loop returns with probability one since at most a quarter of the elements of the multipoint $m[\epsilon; N]$ are roots of P.

In expectation, the procedure yields a point m' which might not be strictly admissible among $m[\epsilon; N]$, but whose value is larger than $\max_i \min\{1, |P(m_i)|^2\}$: Let ρ_0 and ρ_{final} denote the minimum precision required in a deterministic computation of an admissible points and the precision in which the routine is successful. Then, $\max_i |P(m_i)|^2 \geq 2^{-\rho_0}$, and the loop returns a pseudo-admissible point with probability 3/4 as soon as $\rho \geq \rho_0$. Hence, the expected precision demand $E[\rho_{\text{final}}] = \sum_i (1 - \frac{3}{4})^i \cdot 2^i \rho_0 = 2\rho_0$ exceeds the theoretical prediction only by a factor of two.

In practice, we observe that the increase in precision is entirely negligible. Yet, our implementation only models the theoretical result in a Las-Vegas setting.

Heuristics to delay invocations of the Newton step.

The asymptotic cost of the calls to the Newton-Test is dominated by the Taylor shift which is part of the sign-variation test for any interval. However, in practice there is a noticeable impact of unsuccessful Newton-Tests on the performance due to the expensive 01-Tests for the siblings $(a, a^*_{i,j})$ and $(b^*_{i,j}, b)$ of the candidate intervals $(a^*_{i,j}, b^*_{i,j})$. Thus, we strive for calling the Newton-Test only if it will succeed with high likelyhood; vice versa, in the generic situation of well-distributed roots without any distinguished root clusters, as little computation time as possible should be wasted.

We say that a *linear step* on an interval I to I' and I'' leads to a *proper split* if $\mathrm{var}(P, I')$ and $\mathrm{var}(P, I'')$ are both non-zero. In this case, there is at least on root in the neighborhood of I'' that is not part of a potential cluster within I' to be detected by the Newton-Test, and the symmetric argument holds for I''. Hence, a success of the Newton-Test would be an unexpected artifact.

We installed a heuristic to detect such situations and delay Newton-Tests in subtrees with successful bisection steps. To each node in the subdivision tree, We attach a counter that is incremented after each bisection and reset to zero on a proper split. Whenever we examine a node (I, N_I) with $N_I = 4$, we do not try a Newton step until the counter reaches a threshold value of $\log n$ and is reset to zero.[6] For nodes with $N_I > 4$, we always allow calls to the Newton-Test. Thus, we account for situations when there is a distinct cluster of roots, but the resolution N_I is just a little too fine-grained and the candidate interval in the Newton-Test does not comprise the entire cluster.

On intervals with well-separated roots when we never achieve nor profit from quadratic convergence, this heuristic renders it likely that there is never even an attempt to call a Newton-Test, and the overhead of ANewDsc compared to the variant ADsc without the Newton-Test is almost zero. On the other hand, the size of the subdivision tree is increased by at most $O(\log n)$ and, thus, remains polylogarithmic in n and τ in the worst case.

3.2 Changes over the classical RS

Full caching instead of constant-memory version.

The constant-memory strategy in the default instantiation of RS is a prime example of the lazy evaluation paradigm. Instead of caching intermediate results, it favors potentially expensive recomputation from scratch to recover from accumulated round-off errors. Performance comparisons show a clear benefit due to the drastically reduced costs for memory management and RS' sophisticated way of performing incremental computations between adjacent nodes in the subdivision tree [17, Sec. 4].

However, the fast transformations rely on certain structural properties of the subdivision tree: in a pure bisection algorithm, all emerging intervals are of the form $(\frac{c}{2^e}, \frac{c+1}{2^e})$ for some integers c and e. In this setting and with the strong specification of the traversal order of the tree, a good portion of the arithmetic operations for Taylor shifts can be achieved by cheap bitshift and addition operations. In contrast, a general Taylor shift requires significantly more expensive multiplications of arbitrary precision interval values.

These special relations between the intervals is lost when Newton steps are performed or the canonical dyadic subdivision points need to be replaced by admissible points. Optimizations such as the delayed Newton calls are incompatible with strategies that keep only one interval in memory at a time, because the behavior of the algorithm on some interval is no longer independent of the neighborhood: whether a quadratic convergence step is pursued becomes conditional on the outcome of the 01-Test on its sibling.

Fortunately, using the Newton technique, long chains of intervals with a unique descendant in the bisection tree are compressed to only logarithmic height, thus shrinking the entire tree by an order of magnitude. In this light, the impact of the constant-memory strategy is much less pronounced, and it is advisable to switch back to the naive approach of caching all intermediate results. We stress that, without the Newton iteration, the memory consumption in a depth-first traversal would be prohibitive for particular examples with strongly clustered, ill-separated roots such as Mignotte polynomials.

Degree truncation through partial Taylor shifts.

We can consider any univariate polynomial $P(x)$ as in (1) as a function $P(x) = p_n \cdot \prod_{j=1}^{n}(x - \zeta_j)$ in its (not necessarily distinct) complex roots ζ_j. When the domain of interest is restricted to a small region in the complex plane, say, a disk $D \subset \mathbb{C}$, the relative influence of each root ζ_j on P scales with the distance of ζ_j to the points $x \in D$. If there are only $k \ll n$ roots in D and all other roots are well-separated from D, the distance to the remote roots is almost stable for all $x \in D$, and the function $P|_D$ is dominated by the roots in D.

In this situation, the local behavior of P within D is captured in the partial Taylor expansion around the center m up to the k-th term. Hence, there is hope that an approximation of P by its truncated Taylor expansion will suffice for computing isolating intervals for the real roots. Correctness is ensured by conservatively taking into account the truncation error as an interval value of the Lagrangian remainder term. We propose to consider this approach whenever a Newton step succeeds and suggests the existence of a well-separated cluster of $k \ll n$ roots around an interval I. We remark that the multiplicity guess $k := \mathrm{round}(\tilde{k})$ is already computed within the Newton-Test.

Instead of computing $P_I(x)$ as in (2) and the evaluating the 01-Test directly on the full polynomial, we compute an intermediate approximation for $Q_I = P(a + (b-a)x)$ on $I = (a, b)$ as

$$\tilde{Q}_I(x) = \sum\nolimits_{i=0}^{k-1} q_{I,i} x^i + [r]_{I,k} x^k, \quad [r]_{I,k} \supset \frac{P^{(k)}(I)}{k!},$$

where the coefficient $[r]_{I,k}$ of the Lagrangian remainder is evaluated on the entire interval I in conservative interval arithmetic. Now, instead of computing $v_I = \mathrm{var}(P, I)$ on P_I, we perform the 01-Test based on the sign variations $\tilde{v}_I = \mathrm{var}(\tilde{Q}_I, (0,1))$ and $\tilde{v}'_I = \mathrm{var}(\tilde{Q}'_I, (0,1))$.

Lemma. *Let P be a polynomial as in (1) and $I = (a, b)$ and $\tilde{Q}_I$ be as above. Define $\tilde{G}_I := (x+1)^k \, \tilde{Q}_I((x+1)^{-1})$ and $\tilde{H}_I := (x+1)^{k-1} \, \tilde{Q}'_I((x+1)^{-1})$.*

1. *If $\tilde{v}_I := \mathrm{var}(\tilde{G}_I) = \{0\}$, then P has no root in I.*[7]
2. *If $\tilde{v}_I = \{1\}$ and, additionally, $\tilde{v}'_I := \mathrm{var}(\tilde{H}_I) = \{0\}$, then P has exactly one simple real root in I.*

Proof. We consider the polynomials $\tilde{Q}_I$, $\tilde{G}_I$ and $\tilde{H}_I$ simultaneously as polynomials with interval coefficients and

[6] In the notation of [20], *proper splits* occur on a subset of the *special nodes* of the subdivision tree, and we allow paths of $\lceil \log n \rceil$ many *ordinary nodes* before further Newton-Tests.

[7] Recall that the sign variation computation $\mathrm{var}(\cdot)$ on an interval polynomial yields a (super-)set of possible sign variations; see the remarks about the use of interval arithmetic in RS in Section 3.

as the set of exact polynomials with coefficients contained in the intervals. Assume that $\tilde{v}_I = 0$, but P has a root $\zeta = a + \lambda(b-a) \in I$. By the Lagrange representation of the truncation error in Taylor's theorem, there exists a polynomial $F_I = \sum_{i=0}^{k-1} f_I + \frac{P^k(\xi)}{k!} x^k$ of degree k with $F_I(\lambda) = P(\zeta) = 0$ and $\xi \in I$. Descartes' Rule of Signs asserts $\text{var}((x+1)^k F_I((x+1)^{-1})) > 0$. Since $F_I \in \tilde{Q}_I$, it follows that $0 \in \tilde{v}_I$, because the transformations in interval arithmetic overestimate the possible values; a contradiction.

For the second part, an analogous argument on the derivative shows that P is strictly increasing or strictly decreasing on I if $\tilde{v}' = \{0\}$. Since $\tilde{v}_I = \{1\}$, we know that P takes values of different sign at the endpoints of I; hence, P has exactly one root in I. □

If any 01-Test in the processing of a node is unsuccessful, we usually double the working precision. If the local polynomials are computed via a partial Taylor shift, the source of the loss in accuracy can also be the truncation. Hence, before increasing the working precision, we gradually double the multiplicity guess k until we eventually compute the full polynomial. This approach guarantees that, in the worst case, after $\log n$ steps the heuristic falls back to the theoretical model. Hence, even if the heuristic is unsuccessful, the complexity increases at most by a factor of $\log n$.

We observe that partial Taylor shifts give a tremendous speedup for polynomials with tight clusters of low multiplicity. In such situations, we can consider the technique as a partial substitute for the current deficiency of asymptotically fast Taylor shifts. For instances with clusters of constantly many roots, the expected speedup is linear in n: if the heuristic applies, the local polynomials on each active region in the subdivision tree have to computed up to only constantly many coefficients of the Taylor series. This prediction is accurately reflected in the benchmarks.

4. EXPERIMENTS

We present and discuss a short excerpt of our benchmark suite. The objective is to illustrate two main observations: The integration of ANewDsc into RS entails very small overheads for instances with well-distributed roots and shallow subdivision trees, but huge performance gains in challenging situations with clustered roots.

As a baseline, we include the classical version of RS without several optimizations that improve its performance for both classes of inputs, but are incompatible or not yet adapted to the enhanced strategy. For example, we disabled the heuristic that switches to exact arithmetic in certain circumstances because it destroys the guarantees on the expected precision demand, even though it improves the performance for some instances. Also, the hardware floating-point phase of RS is not yet available in ANewDsc. We give three timings for the RS-based solver: besides the classical variant and the full-fledged ANewDsc (denoted as AND in the tables), we show the intermediate version ADsc. It includes the admissible point search and the new subdivision tree layout with full caching, but neither Newton iterations nor degree truncation.

We compare to three well-established competitors:

MPSolve is the leading complex root solver. It is known to keep up with the most efficient real root isolators even though it solves a more general problem. Given our more modest goals, we restrict the region of interest to the real line, and call MPSolve in its latest version 3.1.5 with `unisolve -Gi -SR -Dr -Of -j1 -o1048576`.

CF denotes the continued fraction-based variant of the Descartes method that is available in Mathematica 10. We bypass the high-level interface of the computer algebra system via `System‘Private‘RealRoots` to eliminate the effect of preprocessing stages that are applied by the usual `RootIntervals` call (e.g. the detection and special handling of sparse polynomials or simplifications for even or odd polynomials).

Finally, we compare against Carl Witty's variant of Eigenwillig's bitstream Descartes method in Bernstein basis [6], which constitutes the default real root isolator in the Sage 7.0 open-source computer algebra system. We invoke the algorithm in the optimized variant for 64-bit architectures with `real_roots (f, skip_squarefree=True, wordsize=64)`.

To ensure that all solvers receive sensible instances, we apply a preprocessing stage before running the measurements. It involves the algebraic simplification of even and odd polynomials, computing the square-free part, and factoring out the content of the polynomial. All instances are handled as dense integer polynomials, the fundamental input format for all solvers. The values in the tables are runtimes in seconds, measured on a single run on a server with four 2.5 GHz Intel Xeon E5 CPUs and 256 GB RAM, running Debian 8 64-bit. Degree and coefficient bitsize are denoted by n and τ; an

n	τ	MPS	CF	Sage	RS	ADsc	AND
257	14	0.7	0.1	1.6	7.6	7.7	0.1
513	14	2.9	0.2	2.4	87.6	89.1	0.2
1025	14	13.8	1.1	4.8	*	*	0.7
2049	14	78.5	7.2	12.8			2.9
4097	14	486.3	67.6	85.8			11.2
8193	14	*	224.3	*			43.2
16385	14						188.1
129	128	1.2	0.3	0.5	26.2	25.1	0.1
129	256	2.5	0.9	1.3	100.0	76.9	0.2
129	512	5.9	3.8	3.9	378.5	293.8	0.4
129	1024	16.5	16.7	16.2	*	*	1.3
129	2048	43.4	78.4	*			3.3
129	4096	111.7	431.8				8.1
129	8192	274.7	*				20.8
129	16384	577.7					46.0
129	32768	*					96.8
129	65536						223.9
129	131072						503.0

Table 1: Mignotte: $x^n - ((2^{\tau/2}-1)x-1)^2$

n	τ	MPS	CF	Sage	RS	ADsc	AND
516	140	9.5	21.6	3.3	18.0	18.0	0.8
724	140	38.7	102.1	5.3	73.9	74.9	1.5
1028	140	67.8	565.0	*	230.3	232.4	7.1
1452	140	156.8	*		*	*	5.7
2052	140	283.6					13.1
2900	140	*					24.0
4100	140						88.4
5796	140						175.6
8196	140						394.0
260	160	3.1	1.4	0.3	2.2	2.3	0.9
260	320	4.0	4.6	*call stack overflow*	6.6	5.9	1.8
260	640	5.8	14.9		20.8	19.2	0.9
260	1280	13.2	54.9		77.9	66.4	1.6
260	2560	33.7	222.6		301.2	254.0	3.7
260	5120	98.0	*		*	*	7.7
260	10240	248.0					31.4
260	20480	*					98.5
260	40960						341.2
260	57920						482.4

Table 2: nested Mignotte: $\prod_{i=1}^{4} \left(x^{n/4} - ((2^{\tau/8}-1)x^2-1)^{2i} \right)$

n	τ	MPS	CF	Sage	RS	ADsc	AND
1024	1024	2.5	0.3	4.1	0.6	0.6	0.5
1448	1024	5.0	0.3	4.7	0.5	0.4	0.4
2048	1024	9.8	0.9	8.4	1.6	1.7	1.7
2896	1024	19.2	4.2	19.8	5.7	5.7	5.9
4096	1024	37.5	2.7	29.1	3.8	3.7	3.5
5792	1024	75.2	47.3	60.4	28.6	28.7	32.0
8192	1024	159.4	22.1	183.5	36.0	36.3	36.5
11585	1024	293.2	54.1	$*$	72.3	73.1	73.0
16384	1024	578.9	376.6		275.1	280.9	279.0

Table 3: dense with uniformly random coefficients in $(-2^\tau, 2^\tau) \cap \mathbb{Z}$

n	τ	MPS	CF	Sage	RS	ADsc	AND
180	431	2.3	8.2	1.2	6.5	6.9	0.5
256	506	4.7	34.1	1.6	18.2	19.1	1.1
362	611	10.6	167.4	3.0	57.5	59.4	1.9
512	762	25.9	456.9	3.7	107.7	110.4	3.0
724	973	74.1	$*$	14.7	$*$	$*$	10.2
1024	1274	207.5		22.5			23.6
1448	1696	587.6		55.4			36.9
2048	2296	$*$		$*$			132.2
2896	3143						290.9
4096	4343						596.6

Table 4: clustered: $f^2 - 1$ for $f = \sum_i a_i \binom{n}{i}^{1/2} x^i / \sqrt{i+1}$ with a_i drawn from a normal Gaussian distribution, rounded to $\mathbb{Z}[x]$

asterisk indicates a timeout (> 600 seconds). The full benchmark suite is available on http://anewdsc.mpi-inf.mpg.de/.

Polynomials of Mignotte type are typical benchmark examples that force a large subdivision depth to bisection routines, due to their ill-separated roots at approximately $2^{-\tau} \pm 2^{-n\tau/2}$. We find that the quadratic convergence drastically improves both performance and treesize: The subdivision tree size compresses from approximately 33500 nodes for RS and ADsc to a mere 47 for ANewDsc for $(n, \tau) = (129, 512)$, and the final instance with $\tau = 2^{16}$ leads to a tree with only 65 nodes. For such instances, CF presumably exhibits an almost optimal subdivision tree due to the rational center of the cluster; however, the performance is spoiled by the use of exact arithmetic. The class of nested Mignotte polynomials, shown around an irrational center in Table 2, shows the robustness of ANewDsc both to the higher multiplicity 20 and the irrational center of the cluster.

Random polynomials have a low number ($\Theta(\log n)$) of well-separated real roots; they require only low precision for the isolation and induce flat subdivision trees. Hence, we can expect no gain from the improvements in ANewDsc. On the other hand, the experiments confirm that the enhancements incur almost no additional cost. We observe similar overheads of low constant factors compared to RS on other classes of polynomials with well-distributed roots, such as Wilkinson polynomials with evenly spaced real roots, different classes of orthognal polynomials, or resultants of random bivariate polynomials that arise in projection-based polynomial system solving and algebraic curve analysis.

Finally, polynomials with normal Gaussian-distributed coefficients have a much higher number ($\Theta(\sqrt{n})$) of real roots. By squaring and perturbing such inputs, we construct benchmark instances with many low-multiplicity clusters. We find that ANewDsc quickly detects the clusters, succeeds in the Newton steps, and computes the appropriate number of terms in the partial Taylor shifts.

Acknowledgements

We thank Maplesoft for their support in the development of RS, Leonardo Robol (Katholieke Universiteit Leuven) for his help and support for running the development version of MPSolve and Adam Strzebonski (Wolfram Research) for his instructions on how to control the behavior of the `RootIntervals` function.

5. REFERENCES

[1] G. E. Collins. Continued fraction real root isolation using the Hong bound. *JSC*, 2014.

[2] G. E. Collins & A. G. Akritas. Polynomial real root isolation using Descartes' Rule of Signs. In *SYMSAC*, pp. 272–275, 1976.

[3] Z. Du, V. Sharma & C. Yap. Amortized bounds for root isolation via Sturm sequences. In *SNC*, pp. 113–130, 2007.

[4] Z. Du, V. Sharma & C. K. Yap. *Symbolic-Numeric Computation*, ch. Amortized Bound for Root Isolation via Sturm Sequences, pp. 113–129. Birkhäuser Basel, Basel, 2007.

[5] A. Eigenwillig. *Real Root Isolation for Exact and Approximate Polynomials Using Descartes' Rule of Signs*. PhD thesis, Saarland University, 2008.

[6] A. Eigenwillig et al. A Descartes algorithm for polynomials with bit-stream coefficients. In *CASC*, pp. 138–149, 2005.

[7] A. Eigenwillig, V. Sharma & C. K. Yap. Almost tight complexity bounds for the Descartes method. In *ISSAC*, pp. 151–158, 2006.

[8] I. Z. Emiris, V. Y. Pan & E. P. Tsigaridas. Algebraic algorithms. In *Computing Handbook: Computer Science and Software Engineering*, ch. 10, pp. 1–30. CRC, 3rd ed., 2014.

[9] J. R. Johnson & W. Krandick. Polynomial real root isolation using approximate arithmetic. In *ISSAC*, pp. 225–232, 1997.

[10] W. Krandick. Isolierung reeller Nullstellen von Polynomen. In J. Herzberger, editor, *Wissenschaftliches Rechnen*, pp. 105–154. Akademie Verlag, Berlin, 1995.

[11] K. Mehlhorn & M. Sagraloff. A deterministic Descartes algorithm for real polynomials. *JSC*, 46(1):70–90, 2011.

[12] K. Mehlhorn, M. Sagraloff & P. Wang. From approximate factorization to root isolation with application to cylindrical algebraic decomposition. *JSC*, 66:34–69, 2015.

[13] N. Obreshkoff. *Zeros of Polynomials*. Marina Drinov, Sofia, 2003. Translation of the Bulgarian original.

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

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

[16] N. Revol & F. Rouillier. Motivations for an arbitrary precision interval arithmetic and the MPFI library. *Reliable Computing*, 11(4):275–290, 2005.

[17] F. Rouillier & P. Zimmermann. Efficient isolation of [a] polynomial's real roots. *JCAM*, 162:33–50, 2004.

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

[19] M. Sagraloff. On the complexity of the Descartes method when using approximate arithmetic. *JSC*, 65(0):79–110, 2014.

[20] M. Sagraloff & K. Mehlhorn. Computing real roots of real polynomials. *JSC*, 73:46–86, 2016.

[21] E. P. Tsigaridas. Improved bounds for the CF algorithm. *TCS*, 479:120–126, 2013.

[22] C. K. Yap & M. Sagraloff. A simple but exact and efficient algorithm for complex root isolation. In *ISSAC*, pp. 353–360, 2011.

APPENDIX

A. COMPARISON TO OTHER SOLVERS

Polynomial root solving is a prevailing and fundamental task in numeric and symbolic computation. Many different approaches have been successfully implemented with different objectives. Still, the most successful solvers in widespread use today are variants of only two classes of algorithms: different approaches based on Descartes' Rule of Signs and the Aberth-Ehrlich iteration for complex root finding.

We compared our new method to the state-of-the-art solvers that we are aware of. In the following section, we give a brief survey of the characteristics, weaknesses and strengths of the main candidates for comparison. We do so without any claim for completeness; it is impossible to give due credit to all capabilities of those mature solvers. Also, we ignore methods based on Sturm sequences or Pan's asymptotically near-optimal polynomial factorization, as we are not aware of competitive implementations of those methods.

MPSolve.

Based on the Aberth-Ehrlich root finding iteration, MPSolve stands out in this list as a complex root solver. The numerical method is very efficient in practice, despite the fact that it lacks theoretical convergence guarantees. The computations are done in pure multiprecision arithmetic without interval arithmetic, but the results are certified based on a priori round-off error bounds an a posteriori Gershgorin-type argument.

A major selling point of MPSolve is its triple-stage approach: first, the root isolation is tried with pure hardware floating-point numbers. If this is unsuccessful, a `double` type with extended exponent range is employed, and only if absolutely necessary, multiprecision arithmetic is used. Hence, MPSolve excels for instances where the roots can be isolated within low precision. In such situations, it is on par with dedicated real solvers despite answering a more general problem. Since version 3, another still unique feature of the solver is its full support for multithreading.

Unfortunately, until this date the Aberth-Ehrlich iteration comes without any termination or even worst-case complexity guarantees. In constrast to all other methods in the comparison, the method is intrinsically global; solutions or clusters of roots can be excluded in the later stages of the algorithm, but the boundaries of the region of interest is fuzzy, and nearby roots affect the behavior of the algorithm. Finally, the certification method is extremely sensitive to a very careful theoretical analysis and an accurately matching implementation. For example, it is not immediately clear which modifications are necessary once fast polynomial arithmetic becomes relevant for real-world instances.

For our benchmarks, we use MPSolve in the most recent development version 3.1.5. MPSolve 2.2 is still very slightly superior on selected instances, but both for MPSolve 2.2 and the most recent secsolve algorithm (based on secular equations), we experienced minor bugs in the certification phase. The authors are currently investigating the problems and already corrected some of the issues. For the time being, we only run the unisolve configuration of MPSolve 3 using the call `unisolve -Gi -SR -Dr -Of -j1 -o1048576` on dense inputs with exact integer coefficients. We remark that the `-SR` option restricts the region of interest to the real line, in agreement with our more modest goals.

Continued fraction-based solver in Mathematica.

The implementation in Mathematica is a highly efficient pure real root solver. As RS and the other real solvers, it relies on Descartes' Rule of Signs, but uses a different subdivision strategy: successive computations of root bounds for local polynomials are exploited to compute continued fraction approximations of the roots.

The major benefit of this approach is an extremely fast convergence to rational roots and the choice rational subdivision points with low bitsize of numerator and denominator. For polynomial with only rational roots, the implementation outperforms all other competitors that we are aware of.

However, the price paid in the present variant of the solver is that exact arithmetic is used. Hence, there is no easy way to adapt for bitstream inputs where it can be impossible to decide conclusively whether a subdivision point is a root or not, and termination is not clear without specific safety measures if a root is chosen. Furthermore, even if the algorithm can quickly solve an instance $f \in \mathbb{Z}[x]$, the algorithm can be forced into extremely expensive exact computations on inputs with high bitsize, say $c \cdot f - 1$ for some huge $c \in \mathbb{Z}$, despite the fact that the intrinsic difficulty as a numerical problem remains the same.

The continued fraction scheme achieves optimal convergence against rational roots. However, irrational algebraic numbers are not necessarily easy to approximate in this setting: the golden ratio $\Phi = (1 + \sqrt{5})/2$ is a notoriously hard example and shows the slowest convergence rate possible. Thus, it is easy to construct instances that force continued fraction-based solvers into linear convergence. For example, the Mignotte-like polynomial $x^n - (a\,x - 1)^2$ for an integer $a \in \mathbb{Z}_{>1}$ has roots at approximately $1/a \pm a^{-n/2}$. A subdivision at exactly $1/a$ is easily achieved by the method and a perfect split for the cluster of two roots. However, this is merely an artifact by construction: the polynomial $x^n - (a\,x^2 - 1)^2$ for $a \in \mathbb{Z}_{>1}$ not a square number has roots at approximately $1/\sqrt{a} \pm a^{-n/4}$. But until the algorithm arrives at a sufficiently good rational approximation of the center of the cluster to split the roots, it has to go through through $\Omega(n)$ iterations with only linear convergence.

Those shortcomings are responsible for a loss of one and two orders of magnitudes in the worst-case complexity, and can be easily be enforced by an adversary.

For a fair comparison, we call the continued fraction solver in Mathematica 10 with its default configuration directly via `System‘Private‘RealRoots` to bypass the high-level interface. The latter performs several preprocessing operations, such as detection and optimized handling of sparse polynomials or the replacement of even polynomials $f(x^2)$ by $f(x)$ and reconstruction of the original roots later on.

Sage.

The computer algebra system Sage collects a wealth of high-quality packages and libraries from the open source world in one easily accessible framework, complemented with original implementations of new algorithms. It includes a real solver written in Cython by Carl Witty, which is loosely based on the Bitstream Descartes method by Eigenwillig [5]. The routine by the apt name `real_roots` is unique as a very efficient root finder written in a high-level language, but relies on the well-established MPFI library for interval arithmetic in arbitrary precision. All computations are performed in Bernstein basis using de Casteljau's algorithm, and heuristics for degree reduction are used to achieve similar effects as the partial Taylor shifts in ANewDsc. With respect to the convergence speed, the implementation notes state that a hardware-oriented strategy is used for the selection of the subdivision points. To the best of our knowledge, however, there is no thorough description or analysis of worst or expected case complexity of the algorithm.

Sage's `real_roots` is a very good general-purpose solver. Due to the extensive use of approximate arithmetic, it stays close to the optimum precision demand and is conceptually suited to be used as a bitstream solver. In our experiments, `real_roots` converged quickly to tight clusters. Nevertheless, such situations seem to be its Achilles' heel: In contrast to the competitors, `real_roots` is implemented in a recursive rather than iterative fashion. For inputs that force a very deep computation tree, the algorithm is prone to call stack overflows when there are deeply nested clusters and no tail call elimination applies.

We call solver in the current Sage version 7.0, using the default settings without squarefree decomposition as a preprocessing step, via `real_roots (f, skip_squarefree=True, wordsize=64)`.

Classical RS in Maple.

The classical RS version has already been described before. It serves as a general-purpose solver in the Maple computer algebra system. Over the course of fifteen years of development, a collection of many optimizations and heuristics have been integrated to suit the requirements both as an internal routine in Maple and for direct users.

Some of these improvements are incompatible or significantly less effective when combined with ANewDsc. Others sacrifice the theoretical optimality or are not suited for bitstream inputs; most prominently, this applies to the heuristics for switching to exact arithmetic. Despite the fact that exact computations are only selectively used, the classical RS version can be forced into asymptotically bad behavior with high bitsize inputs in a similar manner as the solver in Mathematica. Hence, the baseline for our experiments is not the version inside Maple, but rather a simplified variant that focuses on simplicity and provably asymptotic optimality. In particular, so far none of the routines for ANewDsc are for interval arithmetic with hardware floating point numbers.

The loss in performance of the stripped-down version is often negligible, but can raise up to a factor of 10 for very particular instances with low intrinsic difficulty such as Wilkinson polynomials, where all roots are real and well-structured, exact arithmetic is asymptotically optimal with respect to the precision demand, and optimizations apply to greatest extent.

B. BENCHMARK SUITE

All our measurements are performed on a server with four Intel Xeon E5-2680 v3 processors with twelve cores each, clocked at 2.5 GHz. The CPU has 30 MB shared L3 cache, and each core has 256 KB L2 cache. The server has a total of 256 GB RAM and runs on Debian 8 "jessie" 64-bit. The measurements are taken from a single run with a timeout of 10 minutes; for timings above 0.1 seconds, we experienced a very stable timing with deviations between runs below one percent for all solvers. However, since the running times are not averaged over many runs, the results on random inputs are only comparable horizontally, not vertically: the solvers are called on the same polynomials, but only one instance for each magnitude is considered. All polynomials are passed as dense polynomials with arbitrary precision integer coefficients. The parsing time for the input is not factored out: we noticed that the parsing in our preliminary interface for ANewDsc is very slow, and for random high-bitsize inputs can take significantly longer than the total time to completion for other solvers. For the time being, the instances are of moderate size such that reading the input is almost negligible even for the variants of RS.

Our full benchmark suite is available on http://anewdsc.mpi-inf.mpg.de/. The readers are invited to verify our results, include their favorite alternative solvers, and inform us about meaningful extensions. We hope that the collection of polynomials grows to a representative corpus of inputs to put polynomial root isolators to the acid test, and look forward to suggestions of both artificial and realistic instances.