# Experimental Evaluation and Cross-Benchmarking of Univariate Real Solvers


Michael Hemmer
Max-Planck-Institut für Informatik
Saarbrücken, Germany

Elias P. Tsigaridas
INRIA Méditerranée
Sophia-Antipolis, France

Zafeirakis Zafeirakopoulos
RISC
Linz, Austria

Ioannis Z. Emiris
University of Athens
Athens, Greece

Menelaos I. Karavelas
University of Crete & FO.R.T.H.
Heraklion, Greece

Bernard Mourrain
INRIA Méditerranée
Sophia-Antipolis, France



## ABSTRACT

Real solving of univariate polynomials is a fundamental problem with several important applications. This paper is focused on the comparison of black-box implementations of state-of-the-art algorithms for isolating real roots of univariate polynomials over the integers. We have tested 9 different implementations based on symbolic-numeric methods, Sturm sequences, Continued Fractions and Descartes' rule of sign. The methods under consideration were developed at the GALAAD group at `INRIA`, the VEGAS group at `LORIA` and the `MPI`-Saarbrücken. We compared their sensitivity with respect to various aspects such as degree, bitsize or root separation of the input polynomials. Our datasets consist of 5 000 polynomials from many different settings, which have maximum coefficient bitsize up to bits 8 000, and the total running time of the experiments was about 50 hours. Thereby, all implementations of the theoretically exact methods always provided correct results throughout this extensive study. For each scenario we identify the currently most adequate method, and we point to weaknesses in each approach, which should lead to further improvements. Our results indicate that there is no "best method" overall, but one can say that for most instances the solvers based on Continued Fractions are among the best methods. To the best of our knowledge, this is the largest number of tests for univariate real solving up to date.



**Categories and Subject Descriptors:** I.1.2 [Symbolic and Algebraic Manipulation]: Algorithms—*Algebraic algorithms*; D.2.8 [Software Engineering]: Metrics—*performance measures*

**General Terms:** Experimentation

**Keywords:** benchmarks, univariate polynomial, real root isolation




*SNC'09,* August 3–5, 2009, Kyoto, Japan.



## 1. INTRODUCTION

Real solving of univariate polynomials is one of the most fundamental research problems, the application range of which touches almost all research areas. Moreover, it is one of the key ingredients for solving polynomial systems. This paper is focused on the comparison of black-box implementations of state-of-the-art algorithms for isolating real roots of univariate polynomials over the integers, that is, the output is supposed to consists of intervals with rational endpoints, each containing exactly one real root of the polynomial

We consider two classes of algorithms for real root isolation of integer polynomials. The first class consists of the subdivision algorithms [7, 11, 23, 27, 8, 29, 16], which exploit either Sturm's theorem or Descartes' rule of signs. The second class contains the Continued Fraction algorithms [1, 32, 30], which are based on the continued fraction expansion of the roots of the polynomial. The best worst case complexity bound for all these algorithms, after eliminating the (poly)logarithmic factors, is $\widetilde{\mathcal{O}}_B(d^4\tau^2)$, where $d$ is the degree of the polynomial and $\tau$ the maximum coefficient bitsize. From a theoretical point of view, the goal is to propose algorithms with complexity bounds that are close to, or match, the bound of the nearly optimal *numerical* algorithm of Pan [28]. The worst case bound of the latter is $\widetilde{\mathcal{O}}_B(d^3\tau)$, which can be further improved to $\widetilde{\mathcal{O}}_B(d^2\tau)$, using sophisticated splitting techniques.

However, in practice, things are quite different. There is no implementation of the optimal numerical algorithm [28], since it is very complicated. Moreover, the known efficient implementations of numerical algorithms [5] could only certify their output using accuracy equal to the theoretical separation bound, that is, the smallest distance between two (possible complex) roots of the polynomial. This makes them quite inefficient. Moreover, with the exception of random polynomials, exact algorithms are more efficient [20, 29, 32, 3, 22, 24] and evidently they can certify their results in all cases. There is also a trend for symbolic-numeric algorithms, which are a combination of both approaches. These algorithms, roughly speaking, work with approximate, actually multi-precision, arithmetic, and if they can not certify their result, then they increase the precision. We refer the reader to [29, 11, 6] and references therein, for more details. However, the complexity of these algorithms is the same as the complexity of their corresponding exact versions. Thus, the main motivation for this work was to evaluate and compare these algorithms experimentally.

Another motivation was the recent need to develop software that exactly handles complex geometric objects. Moreover, our work was facilitated by the fact that `CGAL`[1], the *Computational Geometry Algorithms Library*, is aiming for an algebraic kernel [4]. The univariate part basically consists of three major components: (i) a support for polynomials covering fundamental methods such as GCD computation or square free factorization, (ii) a solver for real root isolation, and (iii) a proper handling of algebraic real numbers, that is, their comparison, approximation and refinement. Our work is the serious evaluation of the different options for the second component, namely the real root isolation.

Previous work includes [21] and [24]. The former describes benchmarks between a solver based on Sturm sequences and the Descartes-based solvers developed at the `MPI`-Saarbrücken. The major conclusion for real root isolation was that for degrees $\geq 20$ the latter are faster or much faster. In [24], the authors introduced the `CGAL`-based univariate algebraic kernel of the VEGAS group, and presented experiments on real root isolation and computation of arrangements of $x$-monotone polynomial curves. For real solving, besides their solver, they also considered the Descartes-based solver of `MPI`, and the `NCF` implementation of GALAAD's kernel. The presented experiments are of two kind: (i) degree 12 polynomials, with only real roots and varying bitsize, and random polynomials of degree 100, and (ii) random polynomials with fixed bitsize (32 and 100) and varying degree (up to 2 000), and Mignotte polynomials. Even though the focus of this work was on the capabilities of the proposed kernel, their conclusions on their set of solvers are similar to ours. In this work we perform experiments with 6 additional solvers, and perform experiments on a richer variety of datasets.

We have tested 9 different implementation that are based on symbolic-numeric methods, Sturm sequences, Continued Fractions and Descartes' rule of sign. The methods under consideration were developed at the GALAAD group at `INRIA`, the VEGAS group at `LORIA` and the `MPI`-Saarbrücken. We compared their sensitivity with respect to various aspects such as degree, bitsize or root separation of the input polynomials. Our datasets consist of 5 000 polynomials from many different settings, which have maximum coefficient bitsize up to bits 8 000, and the total running time of the experiments was about 50 hours. Thereby, all implementations of the theoretically exact methods always provided correct results throughout this extensive study. All results are accessible through a web interface[2] that provides tables and graphs with respect to user specified parameters. To the best of our knowledge, this is the largest number of tests for univariate real solving up to date.

Our results indicate that there is no "best method" overall, but one can say that for most instances the solvers based on Continued Fractions are among the best methods. For each scenario we identify the currently most adequate method, and we point to weaknesses in each approach, which should lead to further improvements, see also Section 5.

The remaining part of the paper is structured as follows. In Section 2 we discuss the different kernels and the investigated root solvers, in particular, we state the known complexity bounds for these methods. In Section 3 we describe the setup of the benchmarks and the used datasets. Section 4 presents the results of the benchmarks and an analysis with respect to certain parameters. In Section 6 we propose possible directions for future work in univariate real solving.

[1] `http://www.cgal.org/`

[2] `http://erga.di.uoa.gr/soft/zaf/unibench/index.html`

# 2. ALGEBRAIC KERNELS

In analogy to the three developed algebraic kernels this section is split into three parts. Each part will give insights about interesting design aspects of the corresponding kernel. For instance, we discuss the implementation of the square free factorization, which is needed to determine the multiplicity of the roots. However, the main focus is on the provided root isolation methods. In particular, we state the known theoretical complexity bounds of the methods.

In what follows $\mathcal{O}_B$-notation refers to bit complexity and the $\widetilde{\mathcal{O}}_B$-notation means that we are ignoring (poly-)logarithmic factors. Finally, in all cases, $d$ will be the degree of the polynomials and $\tau$ the maximum coefficient bitsize.

## 2.1 The GALAAD Kernel

The algebraic kernel by `INRIA` relies on SYNAPS, which in turn is now provided by the package `Realroot` [3] of `mathemagix`[4]. The `mathemagix` project is an open source effort that provides fundamental algebraic operations such as algebraic number manipulation tools, different types of univariate and multivariate polynomial real root isolation methods, resultant and GCD computations, etc. The main motivation behind this project, is the need to combine symbolic and numeric computations, which is ubiquitous in many problems. We refer the reader to [26] for more details.

The library provides 4 exact methods for real root isolation of univariate polynomial with integer coefficients, namely `Sturm`, `CF`, `NCFF` and `NCF`. Moreover, we investigated an approximative solver `Sleeve`, who's computed intervals are not necessarily isolating. And an experimental implementation of a hybrid `Symbnum` that combines `Sleeve` with the `CF` method. For a discussion see the subsequent subsections.

Compared to the other kernels, a principal difference is that most solvers in this library exploit specialized algorithms for polynomials of degree up to 4. Hereafter, we refer to these algorithms as `IDS` (Isolation and Discrimination Systems). These systems are based on pre-computed Sturm sequences and they compute rational points that isolate the real roots, as functions in the coefficients of the polynomial. The arithmetic complexity of these methods is $\widetilde{\mathcal{O}}(1)$, whereas the bit-complexity is $\widetilde{\mathcal{O}}_B(\tau)$, see [17, 31] for more details. The algorithms that use `IDS` methods are `Sturm`, `Sleeve`, `Symbnum` and `CF`. The solvers `NCFF` and `NCF` do not apply `IDS`.

### 2.1.1 *Sturm*

A subdivision method based on Sturm's theorem. We refer to it as `ST` in the figures. The algorithm employs polynomial remainder sequences (`PRS`) as a real root counting query. In order to determine whether a root is isolated by a certain interval, the `PRS` is evaluated at the endpoints of the interval at question. Traditionally, it is expected to be the slowest method. The major disadvantage of `Sturm` is that it must

[3] http://www-sop.inria.fr/galaad/mathemagix/realroot

[4] http://www.mathemagix.org/

compute the full PRS before it can even start to isolate the roots. Moreover, the subsequent evaluation of the PRS is in general more complex than other methods. Thus, we use the method primarily as a reference. The complexity of the algorithm is $\widetilde{\mathcal{O}}_B(d^6 + d^4\tau^2)$, see also [8, 16, 9] and references therein.

#### 2.1.2 CF-Family

We investigated three solvers based on the Continued Fractions algorithm (CF) algorithm. The CF algorithm computes the continued fraction expansion of the real roots of the polynomial to compute isolating interval for them. One of its main ingredients is the computation of lower bounds on the positive real roots. Different methods of computing lower bounds, lead to different variants of the algorithm. However, all investigated solvers in this article use the same algorithm to compute lower bounds on the roots, namely a modification of Hong's bound proposed in [1]. We refer the reader to [30, 32, 1] and references therein, for a detailed description. The method is known to be among the most powerful root isolation approaches, in particular, for ill-conditioned problems. A disadvantage is that the method uses exact arithmetic throughout the algorithm, that is, in terms of its bit-complexity the algorithm is not adaptive. The worst case complexity of the algorithm is $\widetilde{\mathcal{O}}_B(d^5\tau^2)$ [30], while the average case complexity is $\widetilde{\mathcal{O}}_B(d^4 + d^3\tau)$ [32] , assuming that the Gauss-Kuzmin distribution holds for the real algebraic numbers.

The three investigated solvers are CF, NCFF, and NCF.

CF This solver is an implementation of the CF algorithm that does not use any external factorization tools. The square free factorization is provided by the package realroot of mathemagix. In case the polynomial degree is $\leq 4$ the method employs IDS.

NCFF This method is based on the same algorithm, but the implementation employs the NTL library for square free factorization and factorization over the integers. Integer arithmetic in NTL is based on GMP, which is used through a SYNAPS-NTL conversions interface, see [14]. NCFF does not apply IDS.

NCF The difference between NCFF and NCF is that NCF employs square free factorization, but not factorization over the integers. NCF does not apply IDS.

#### 2.1.3 Sleeve and Symbnum

Sleeve is an approximate real root isolation method that uses double arithmetic. We refer to it as SV in the figures. The method computes an "upper" and "lower" (i.e. a sleeve) approximation of the polynomial. A Descartes-like subdivision algorithm is applied to the sleeve. The approximation is refined, if possible, until the machine precision is reached. The algorithm is certified in the sense that it can not miss roots. However, an interval may contain more than one root. In some cases we can certify the result using a sign test with the first derivative and interval arithmetic, that is, we consider the method as a possible filter. In case the polynomial degree is $\leq 4$ the method employs IDS.

Symbnum is an experimental symbolic-numeric algorithm, which is a combination of the Sleeve and the Continued Fraction algorithm. We refer to it as SN in the figures. Initially the Sleeve algorithm runs and produces some intervals, that may contain more than one real root. In the sequel, using exact arithmetic, we compute the continued fractions expansion of the real root(s) that are in the interval, the resulting interval is transformed to (0, 1) and the Sleeve algorithm is applied again (possible after scaling the coefficients). In case the polynomial degree is $\leq 4$ the method employs IDS.

For the solvers using polynomials represented in the Bernstein basis (eg. "Sleeve" and "Symbnum"), the conversion to approximate arithmetic is done via a basis conversion and scaling using exact (rational) arithmetic and then rounding the coefficients up and down to nearest approximate numbers. When the bit size of the input is increasing this conversion is becoming significative.

Although the approximate solvers in some cases do not isolate all the roots, they can however be interesting as filters within a given precision. This depends of course on the applications where you are using these solvers, for instance if the input is not "exact".

## 2.2 The MPI Kernel

The kernel provided by the MPI is developed within CGAL and follows the *generic programming paradigm* [2] using the template technique of C++. This allows the exchange of the representation of the algebraic real roots as well as the real root isolation method, which also selects the used coefficient type. That is, the kernel can, potentially, combine the best root isolator with the best representation of algebraic reals due to its generic design. In the context of this work the most important template argument is the second, namely the root isolator. In order to be a valid template argument the interface of the isolator class must meet a few simple requirements which are gathered in a so-called concept, for an exact definition of this concept we refer the reader to [19].

Up to date, the MPI has developed two Descartes-based root isolators, namely DSC and BSDSC. A major advantage of both implementations is the possibility to exchange the coefficient type. In particular, the BSDSC is intended to work over algebraic extensions, as it is reported in [19, 10, 11], for example, it can isolate roots of $f(x, y)_{|x=\alpha}$, where $\alpha$ is an algebraic number.

By the time of the benchmarks the MPI kernel did not employ modular arithmetic in order to compute the GCD, which is used in the square-free factorization. During the benchmarks, this proved to be a serious drawback in some of the test cases. However, the two methods provided by the MPI kernel will benefit from the recent integration of a modular GCD [18] into the polynomial package of CGAL.

#### 2.2.1 DSC

A real root isolation method based on Descartes' rule of signs. The solver is integrated in CGAL and implemented according to [7]. A major disadvantage is that the method uses exact arithmetic throughout the algorithm, that is, in terms of its bit-complexity the algorithm does not adapt to the hardness of the particular isolation problem. Traditionally, it is expected to be slower than the other methods. Thus, we use the method primarily as a reference. Given the new tree bound in [12, 10] the complexity of the algorithm is $\widetilde{\mathcal{O}}_B(d^4\tau^2)$.

### 2.2.2 BSDSC

The method is named Bitstream-Descartes due to the fact that the coefficients of the polynomial are converted to (potentially infinite) bitstreams. A variant of the Descartes method is used to find isolating intervals for the real roots. The advantage of this method is its adaptiveness in terms of bit-complexity, see also [11, 10]. However, since the method overestimates the number of required bits, it can happen that the bitstreams become larger than the actual coefficients. In this case the method may even become more costly than the pure Descartes algorithm `DSC`. According to [10], the worst case complexity of `BSDSC` is $\widetilde{\mathcal{O}}_B(d^5\tau^2)$.

## 2.3 The VEGAS Kernel

The kernel which is developed by the VEGAS group at `LORIA` is based on the `RS` library[5], which in turn is developed by the SALSA group at `INRIA`-Rocquencourt, see [24, 29]. The kernel is dedicated for one particular coefficient type, namely the GMP arbitrary-length integers. In the square free factorization the kernel uses its own modular GCD implementation. For root isolation the kernel interfaces the solver provided by `RS`.

### 2.3.1 RS

The `RS` solver is based on the interval Descartes algorithm [6]. The algorithm uses multi-precision floating point arithmetic to convert the coefficients to intervals of a certain initial precision. Thereafter, it tries to apply Descartes' rule of signs using interval arithmetic. In case this is not applicable the algorithm increases the precision of the approximations. In order to improve the memory usage of the method, the implementation takes special care about the order of the transformations that are required for the application of Descartes' rule of signs. The advantage of this method is its adaptiveness in terms of bit-complexity.

# 3. DATASETS

Our data sets cover classical benchmark instances such as Mignotte polynomials or random polynomials as well as instances motivated by geometric applications. In order to reveal the advantages or disadvantages of the solvers with respect to certain characteristics of the input polynomials, each data set tries to focus on a certain aspect. In total, our benchmarks use 5 200 polynomials, distributed over 155 datasets:

`[rnd]` datasets contain polynomials with random integers as coefficients. The degree of the polynomials ranges from 3 to 100, whereas the bitsize of the coefficients ranges from 10 to 50, which is considered to be rather small. Consequently, these polynomials are the easiest to solve among the ones tested. However, random polynomials are important, since they occur naturally in many problems.

`[bts]` datasets are intended to test the sensitivity of the methods with respect to the coefficient length. The datasets contain polynomials with random coefficients but with significantly larger bitsize. The `[bts]` datasets contain polynomials of degree between 3 and 100 and bitsize in the range from 2 000 to 8 000. Note that the polynomials in this dataset are not particularly harder to solve than those contained in `[rnd]`. However, methods that are not adaptive are expected to be more afflicted by the increased bitsize.

`[3darr]` are datasets that have already been used in [19]. The polynomials are motivated by computations of arrangements in 3D. Each dataset contains 100 univariate polynomials of bitsize 200, 400, 600, 800, 1 000, 1 200, 1 400, 1 600, 1 800 or 2 000 and degree 12. Each polynomial is the product of six parabolas, where each parabola has at least one root within the interval $[-10, 10]$. Consequently, all polynomials have 12 real roots.

`[vorell]` is a dataset that was generated by a concrete geometric application, namely the exact computation of the Voronoi diagram of ellipses [13, 15]. The polynomials are resultants that describe all tritangent circles to three ellipses. The dataset contains 10 polynomials of degree 184. However, each polynomial has only 8 real roots. The bitsize of each coefficient is, approximately, 3 500 bits.

`[int]`/`[rat]` datasets contain polynomials with integer and rational roots, respectively. While the roots for the construction of the polynomials are chosen uniformly at random, there are cases with multiple selections of the same root. (e.g. small bitsize, high degree). The datasets consist of combinations of degree from 3 to 100 and bitsize from 10 to 50. These datasets provide some insight concerning the behavior of the 9 methods in "easy" problems.

`[mgn]` datasets contain Mignotte polynomials [25], that is polynomials of the form $x^d - 2(kx-1)^2$, of degree between 3 and 100 and bitsize from 10 to 50. The `[mgn]` polynomials achieve very small real root separation, having 2 real roots very close to each other. Usually, the real root separation is not known a priori, thus it is important to know how a method behaves in situations of small real root separation.

All datasets used during the benchmarks are available at `http://erga.di.uoa.gr/soft/zaf/unibench/index.html`.

# 4. RESULTS AND DISCUSSION

The benchmarks took place on a 32-bit Pentium III with 256MB RAM memory, Debian 4.0 GNU/Linux. Compilation was done using `g++` version 4.1.2 with optimization flags `-O3` and `-DNDEBUG`. We used the internal release 271 of `CGAL` (released 03-Apr-2008), as well as the software libraries GMP[6](version 4.2) and NTL[7] (version 5.4.1). The `MPI` kernel, was instantiated with the integers provided by the CORE library, which is also maintained by CGAL and also based on GMP. The integer arithmetic of SYNAPS is based on GMP, except for the approximative solvers that naturally rely on usual floating point arithmetic. Moreover, special efficient routines are available for converting to and

[5] `http://fgbrs.lip6.fr`

[6] `http://gmplib.org/`

[7] `http://www.shoup.net/ntl`

from NTL integer types. Time, in msec, was measured using the `clock()` function of the `ctime` library.

For each method we measured the time that was needed to determine all real roots including the multiplicity of these roots, that is, we implicitly tested the implemented square free factorization as well. However, in the case of `[3darr]` and `[mgn]` datasets the polynomials where known to be square free, which was indicated to the solvers by a corresponding flag.

In order to reduce noise, we computed the average time over a number of iterations such that the total time for each polynomial exceeded 1 msec. Since there were only small variances for different instances in the same dataset, the reported time is the average over all polynomials in each dataset. In case a method took more than 30 seconds for some instance or it failed to isolate all the real roots of a polynomial, we ignored the measurement.

In the sequel, we discuss the results with respect to the characteristics of the problems, as well as robustness and effectiveness issues. In the tables, bold entries indicate the fastest method for each case.

## 4.1 Polynomials of low degree

Table 1 presents the timings for polynomials of low degree, which originate from the various datasets. The best times in each row are emphasized. Note that the table does not list `Sturm`, `Sleeve`, `Symbnum` and `CF`, since all these algorithms apply the same method, namely `IDS`, for polynomials of degree $\leq 4$. In order to achieve higher accuracy in timing, these tests included 100 iterations for each polynomial in the set.

**Table 1.** Polynomials of degree 3.

| bits | IDS | NCFF | NCF | DSC | BSDSC | RS |
|---|---|---|---|---|---|---|
| | | | [rnd] | | | |
| 10 | **0.066** | 0.288 | 0.240 | 0.256 | 0.696 | 0.520 |
| 20 | **0.096** | 0.280 | 0.224 | 0.568 | 0.720 | 0.696 |
| 30 | **0.084** | 0.688 | 0.208 | 0.624 | 0.536 | 0.880 |
| 40 | **0.063** | 0.672 | 0.272 | 0.392 | 0.400 | 0.880 |
| 50 | **0.240** | 0.720 | 0.344 | 0.328 | 1.016 | 1.016 |
| | | | [mgn] | | | |
| 10 | **0.148** | 0.800 | 1.000 | 1.560 | 1.840 | 8.800 |
| 20 | **0.318** | 1.920 | 1.640 | 2.400 | 1.960 | 24.400 |
| 30 | **0.326** | 1.920 | 2.000 | 3.800 | 2.400 | 49.000 |
| 40 | **0.312** | 2.000 | 2.000 | 4.000 | 2.400 | 76.800 |
| 50 | **0.442** | 3.000 | 3.000 | 4.400 | 4.000 | 116.000 |
| | | | [int] | | | |
| 10 | **0.231** | 0.856 | 0.352 | 1.040 | 1.120 | 0.824 |
| 20 | **0.316** | 1.160 | 0.520 | 0.720 | 0.920 | 1.600 |
| 30 | **0.310** | 1.520 | 0.368 | 0.560 | 1.360 | 1.200 |
| 40 | **0.408** | 1.400 | 0.384 | 0.720 | 1.520 | 1.440 |
| 50 | 0.592 | 1.880 | **0.560** | 0.920 | 1.240 | 1.520 |
| | | | [rat] | | | |
| 10 | **0.293** | 0.856 | 0.312 | 0.800 | 0.600 | 1.480 |
| 20 | **0.326** | 1.640 | 0.336 | 1.520 | 0.720 | 1.520 |
| 30 | 0.378 | 1.400 | **0.368** | 0.880 | 1.040 | 1.760 |
| 40 | 0.464 | 1.520 | **0.344** | 1.040 | 1.200 | 1.440 |
| 50 | 0.856 | 1.600 | **0.376** | 1.680 | 0.560 | 1.520 |
| | | | [bts] | | | |
| 2000 | 2.1975 | 1.896 | 1.856 | 0.880 | **0.680** | 1.288 |
| 4000 | 5.342 | 2.112 | **1.400** | 1.424 | 1.560 | 1.976 |
| 6000 | 2.25 | 3.280 | 1.984 | 1.856 | **1.712** | 2.800 |
| 8000 | 5.884 | 4.800 | 2.480 | 1.792 | **1.680** | 3.720 |

For the `[rnd]` instances, random polynomials with small bitsizes, all solvers perform quite well. However, `IDS` methods are clearly the fastest, which is due to the small coefficient size of the input polynomials. For the Mignotte polynomials `[mgn]` the advantage of the `IDS` method is even more evident. This is due to the fact that, in the case of the `IDS` methods, the boundaries of the isolating intervals are in principal computed directly by a formula, whereas the other solvers have to subdivide several times to separate the close roots of the Mignotte polynomials.

In case of polynomials from the `[int]` and `[rat]` datasets, the `NCF` is comparable or even faster than the `IDS` methods. This is caused by the fact that, in the case of integer/rational roots, the lower bound that is computed by the CF algorithm is tight, that is, it equals the roots and the subdivision stops immediately. The `NCFF` method can not take advantage from this fact due to the additional overhead of the factorization procedure. `BSDSC` and `RS` can not detect these special roots since both algorithms work with approximated coefficients.

For random polynomials with moderate bitsize (`[bts]`) the `BSDSC` is finally faster than the other methods. The reason is that it only uses a few leading bits of the polynomials which is enough to isolate and certify the roots. Up to 4 000 bits, the `NCF` method is also comparable. `IDS` is not competitive due to its non-adaptive implementation.

## 4.2 Polynomials of small real root separation

s The `[mgn]` datasets were incorporated into these benchmarks in order to study the behavior of the methods on inputs with small root separation. This is important, since the real root separation is usually not known a priori. In particular, the `[mgn]` instances force the subdivision based solvers to reach their worst case complexity.

The results are given in Table 2. Note that the table reports only one column for the `CF`-Family. This is due the fact that the `[mgn]` instances are known to be square free, which was indicated by a flag. Thus, the implementations for `CF`, `NCFF` and `NCF` can be considered as identical since they only differ in the way the square free factorization is implemented.

For the `[mgn]` instances the `CF`-Family is clearly the fastest method. Even for moderate degrees, only the `CF`-Family is capable of solving Mignotte polynomials in reasonable time. For example, the times are not even comparable for `[mgn]` of degree 30. The only method that is comparable up to degree 10 is `BSDSC` and even though the root separation is very small it is still faster than `DSC`. The `RS` solver performs seriously worse, in particular, it is even slower than `DSC`. The problem is due to the high memory consumption of the algorithm. In cases of degree higher than 30 the method would run out of memory. As expected `Sturm` is the slowest method.

Table 2 does not report on `Sleeve` and `Symbnum` since in datasets of higher degrees or bitsizes both methods took much longer than the other methods or even fail to isolate all the real roots correctly. This is due to the use of inexact double arithmetic within `Sleeve`.

## 4.3 Polynomials of big bitsize

The `[bts]` instances are incorporated in order to reveal the dependency on the bitsize of the input. Note that the polynomials are still random, that is, in terms of root separation they have the same behavior as the `[rnd]` instances. The reported times cover the time spent to detect that the polynomials are square free as well as the root isolation it self. This also explains the difference in the `CF`-Family. Table 3 shows the results.

**Table 2.** `[mgn]` polynomials.

| bits | Sturm | CF-Fam. | DSC | BSDSC | RS |
|---|---|---|---|---|---|
| | | [mgn]: Degree 5 | | | |
| 10 | 3.2 | **1.6** | 1.8 | 2.0 | 6.8 |
| 20 | 7.6 | **2.0** | 2.0 | 2.2 | 15.2 |
| 30 | 10.8 | **3.0** | 4.0 | 3.6 | 26.0 |
| 40 | 15.8 | **3.6** | 4.2 | 4.2 | 40.8 |
| 50 | 21.4 | 5.0 | 6.0 | **4.4** | 57.2 |
| | | [mgn]: Degree 10 | | | |
| 10 | 7.6 | **1.8** | 4.0 | 4.0 | 11.8 |
| 20 | 19.6 | **3.6** | 7.8 | 8.0 | 26.4 |
| 30 | 33.4 | **4.8** | 12.4 | 12.0 | 43.6 |
| 40 | 52.4 | **7.6** | 17.6 | 15.8 | 68.0 |
| 50 | 77.6 | **9.4** | 24.6 | 19.8 | 95.2 |
| | | [mgn]: Degree 30 | | | |
| 10 | 80.4 | **4.4** | 76.2 | 76.6 | 96.4 |
| 20 | 321.7 | **11.7** | 242.4 | 187.9 | 285.5 |
| 30 | 868.8 | **19.6** | 504.1 | 357.1 | 633.6 |
| 40 | 1866.1 | **29.2** | 854.0 | 571.2 | 1127.9 |
| 50 | 3345.3 | **40.1** | 1295.8 | 783.9 | 1780.6 |
| | | [mgn]: Degree 50 | | | |
| 10 | 425.0 | **10.0** | 627.0 | 486.8 | 493.4 |
| 20 | 2444.0 | **28.4** | 2305.2 | 1582.4 | 2700.0 |
| 30 | 7602.0 | **51.4** | 5230.8 | 3206.4 | 6386.8 |
| 40 | 15992.2 | **78.6** | 12054.0 | 5861.4 | 11443.2 |
| 50 | 29494.6 | **118.4** | . | . | 18624.8 |
| | | [mgn]: Degree 100 | | | |
| 10 | 7882.6 | **32.4** | 22486.8 | 14096.0 | 23852.8 |
| 20 | . | **121.4** | . | . | . |
| 30 | . | **251.4** | . | . | . |
| 40 | . | **429.6** | . | . | . |
| 50 | . | **674.8** | . | . | . |

**Table 3.** `[bts]` polynomials.

| bits | Sleeve | Symbnum | CF | NCFF | NCF | DSC | BSDSC | RS |
|---|---|---|---|---|---|---|---|---|
| | | | [bts]: Degree 3 | | | | | |
| 2000 | 2.87 | 2.91 | 3.01 | 1.90 | 1.86 | 0.88 | **0.68** | 1.29 |
| 4000 | 7.10 | 7.09 | 7.17 | 2.11 | 1.40 | **1.42** | 1.56 | 1.98 |
| 6000 | 2.88 | 3.12 | 3.00 | 3.28 | 1.98 | 1.86 | **1.71** | 2.80 |
| 8000 | 7.82 | 7.85 | 7.86 | 4.80 | 2.48 | 1.79 | **1.68** | 3.72 |
| | | | [bts]: Degree 10 | | | | | |
| 2000 | 3.56 | 3.24 | **1.30** | 2.34 | 1.32 | 1.73 | 1.90 | 2.42 |
| 4000 | 7.72 | 7.64 | **1.57** | 3.84 | 2.01 | 1.94 | 2.11 | 3.24 |
| 6000 | 13.4 | 13.3 | **1.62** | 5.84 | 2.20 | 2.56 | 2.31 | 3.80 |
| 8000 | 19.6 | 19.6 | **1.95** | 7.92 | 2.84 | 2.40 | 2.16 | 4.52 |
| | | | [bts]: Degree 30 | | | | | |
| 2000 | 10.9 | 10.7 | 2.9 | 9.4 | **2.5** | 4.7 | 6.1 | 5.8 |
| 4000 | 24.2 | 24.2 | **4.4** | 19.9 | 4.5 | 7.1 | 6.8 | 7.9 |
| 6000 | 39.5 | 39.2 | **4.8** | 64.4 | 5.6 | 9.0 | 6.9 | 9.6 |
| 8000 | 59.0 | 59.2 | **6.1** | 24.2 | 6.9 | 10.9 | 7.2 | 11.6 |
| | | | [bts]: Degree 50 | | | | | |
| 2000 | 19.8 | 19.7 | 7.5 | 36.1 | **5.8** | 11.6 | 15.2 | 11.8 |
| 4000 | 40.7 | 40.7 | 10.2 | 52.1 | **8.5** | 17.3 | 15.6 | 14.8 |
| 6000 | 67.9 | 67.7 | 13.1 | 112.9 | **11.4** | 21.2 | 16.7 | 18.4 |
| 8000 | 101.9 | 102.1 | 19.0 | 126.6 | **16.2** | 29.3 | 17.5 | 23.9 |
| | | | [bts]: Degree 70 | | | | | |
| 2000 | 28.0 | 28.3 | 11.7 | 74.3 | **8.7** | 19.8 | 29.2 | 17.5 |
| 4000 | 59.4 | 59.4 | 20.3 | 88.5 | **15.9** | 32.2 | 30.7 | 24.8 |
| 6000 | 97.3 | 97.4 | 23.9 | 145.8 | **19.7** | 41.5 | 31.3 | 28.8 |
| 8000 | 144.8 | 145.0 | 31.2 | 179.0 | **25.5** | 52.3 | 32.0 | 35.0 |
| | | | [bts]: Degree 100 | | | | | |
| 2000 | 43.6 | 43.4 | 25.4 | 160.5 | **17.9** | 44.3 | 65.9 | 34.9 |
| 4000 | 88.1 | 88.1 | 36.7 | 179.2 | **28.5** | 63.5 | 66.9 | 42.2 |
| 6000 | 144.2 | 144.6 | 53.5 | 239.8 | **41.6** | 90.4 | 68.1 | 53.8 |
| 8000 | 215.5 | 215.2 | 72.1 | 551.6 | **57.8** | 114.5 | 70.4 | 66.6 |

For polynomials of small degree the Descartes solvers are remarkably fast, in particular they are even faster than `CF`, which uses `IDS` for degrees $\leq 4$. For moderate degrees the `CF` method is faster, whereas the `NCF` method is the fastest for degrees $\geq 50$. The difference can be explained by the overhead that `NCF` has to pay for the conversion to and from NTL. For high degrees this pays off due to the more efficient implementation of polynomial's operations in NTL. The `NCFF` method is the slowest method due to the extra time spent in the, in this case, useless factorization.

Besides `NCFF`, the slowest methods are `Sleeve` and `Symbnum`. Actually, this is a surprise since these approximation methods should be able to solve these easy polynomials fast. The behavior of `RS` is similar to `CF` and `NCF`, but has a larger constant overhead.

The `Sturm` method is not reported here since it was not competitive at all. This is due to the huge coefficient growth in the intermediate terms of the polynomial remainder sequence that must be computed by `Sturm`.

**Figure 1.** `[bts]` polynomials of degree 50

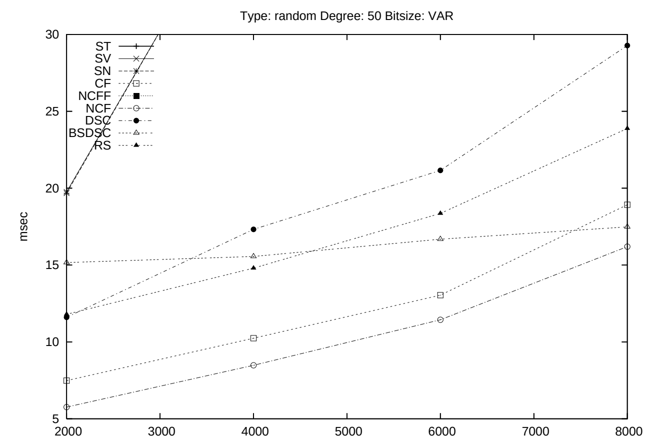
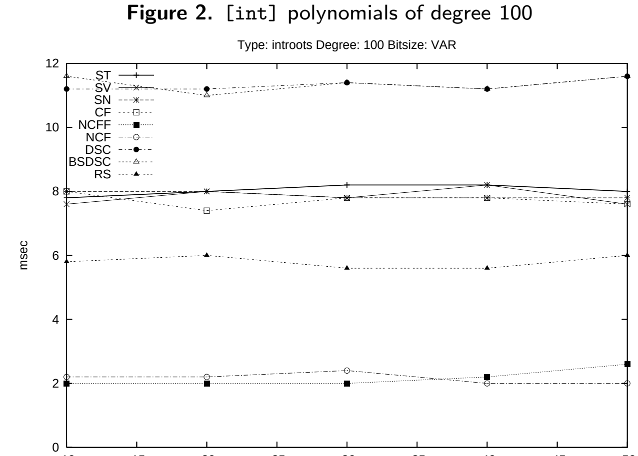



Figure 1 illustrates the behavior for polynomials of degree 50, while the bitsize increases. Every method shows almost linear runtime growth with respect to the number of bits. However, the interesting part are the slopes. `Sleeve` and `Symbnum` show the worst behavior, whereas `CF`, `NCF`, `DSC` and `RS` show a moderate increase. The behavior of `BSDSC` is remarkable since it is almost not effected by the increase of the bitsize. However, for polynomials of degree 50 with less than 8 000 bits, we can not observe the break even point for `BSDSC` due to the huge constant overhead. As one can see in Table 3, this constant overhead depends on the degree of the polynomials, which has a serious impact on the number of required bits within `BSDSC`.

## 4.4 Polynomials of integer/rational roots

**Figure 2.** `[int]` polynomials of degree 100




The `[int]` and `[rat]` instances indeed contain polynomi-

**Table 4.** [int] polynomials.

| bits | Sturm | Sleeve | Symbnum | CF | NCFF | NCF | DSC | BSDSC | RS |
|---|---|---|---|---|---|---|---|---|---|
| [int]: Degree 5 | | | | | | | | | |
| 10 | 1.120 | 0.88 | 1.00 | **0.84** | 1.88 | 0.92 | 1.48 | 1.48 | 1.23 |
| 20 | 1.960 | 1.88 | 1.28 | **1.40** | 1.88 | 1.52 | 1.56 | 1.52 | 1.64 |
| 30 | 2.840 | **1.32** | 1.76 | 1.68 | 1.88 | 1.36 | 1.56 | 1.76 | 1.88 |
| 40 | 4.200 | 1.68 | 1.64 | **1.16** | 2.00 | 1.76 | 1.16 | 1.52 | 1.68 |
| 50 | 4.800 | 1.68 | 1.84 | **1.20** | 2.00 | 1.28 | 1.28 | 1.52 | 1.92 |
| [int]: Degree 10 | | | | | | | | | |
| 10 | 1.76 | 1.88 | 1.64 | 1.52 | 1.32 | **1.00** | 1.80 | 2.16 | 1.52 |
| 20 | 1.76 | 1.80 | 1.60 | 1.72 | 1.72 | **1.24** | 1.96 | 2.76 | 1.60 |
| 30 | 3.60 | 2.40 | 2.60 | 2.80 | 2.80 | **1.88** | 2.20 | 3.40 | **1.88** |
| 40 | 7.80 | 3.20 | 3.20 | 3.40 | 3.40 | **2.60** | 3.40 | 4.00 | 3.40 |
| 50 | 13.00 | 4.00 | 3.60 | 4.00 | 4.00 | 3.60 | **3.40** | 3.80 | 3.80 |
| [int]: Degree 30 | | | | | | | | | |
| 10 | 3.34 | 3.02 | 3.18 | 3.20 | **1.77** | 1.58 | 3.72 | 3.46 | 2.01 |
| 20 | 3.30 | 3.00 | 3.00 | 3.12 | 1.68 | **1.63** | 3.76 | 3.74 | 1.98 |
| 30 | 3.28 | 3.10 | 3.22 | 3.04 | 1.65 | **1.50** | 3.60 | 3.66 | 2.09 |
| 40 | 3.14 | 2.90 | 3.02 | 3.08 | 1.69 | **1.63** | 3.66 | 3.76 | 1.99 |
| 50 | 3.20 | 2.90 | 2.94 | 3.16 | 1.67 | **1.65** | 3.54 | 3.68 | 2.12 |
| [int]: Degree 50 | | | | | | | | | |
| 10 | 4.00 | 4.00 | 4.00 | 4.00 | 1.92 | **1.76** | 5.60 | 5.00 | 3.80 |
| 20 | 4.00 | 4.00 | 4.00 | 4.00 | 1.92 | **1.84** | 4.40 | 4.80 | 4.00 |
| 30 | 4.00 | 4.00 | 3.80 | 4.00 | 1.64 | **1.60** | 5.40 | 5.20 | 3.80 |
| 40 | 4.00 | 3.80 | 4.00 | 4.00 | 1.84 | **1.68** | 5.80 | 5.00 | 3.60 |
| 50 | 4.00 | 4.00 | 4.00 | 4.00 | 1.92 | **1.84** | 4.60 | 5.40 | 3.80 |
| [int]: Degree 100 | | | | | | | | | |
| 10 | 7.80 | 7.60 | 8.00 | 8.00 | **2.00** | 2.20 | 11.20 | 11.60 | 5.80 |
| 20 | 8.00 | 8.00 | 8.00 | 7.40 | **2.00** | 2.20 | 11.20 | 11.00 | 6.00 |
| 30 | 8.20 | 7.80 | 7.80 | 7.80 | **2.00** | 2.40 | 11.40 | 11.40 | 5.60 |
| 40 | 8.20 | 8.20 | 7.80 | 7.80 | 2.20 | **2.00** | 11.20 | 11.20 | 5.60 |
| 50 | 8.00 | 7.60 | 7.80 | 7.60 | 2.60 | **2.00** | 11.60 | 11.60 | 6.00 |

als with multiple roots, and are meant to provide an insight on the behavior of the 9 methods in "easy" problems. However, the obtained times are notably effected by the time spent, not by root isolation, but within square-free factorization and factorization (if this operation is available). A combination of the latter two operations could compute the real roots exactly in all the cases. Thus, our results on these datasets are, by no means, conclusive and further experimentation is needed with polynomials of considerably higher degree and bitsize.

In Figure 2, we have 4 bands of methods, grouped by the factorization implementation they employ. The fastest factorization implementation is provided by the NTL library, which is used by `NCFF` and `NCF`. The square-free factorization in the LORIA kernel (`RS`) is a bit faster than the one within **`mathemagix`** (`Sleeve`, `Symbnum`, `CF`). The square free factorization of the MPI kernel (`DSC`, `BSDSC`) is not competitive. This was expected, since the MPI kernel did not apply any modular arithmetic within the square free factorization by the time of these benchmarks. However, the `MPI` kernel will benefit from the recent integration of a modular GCD into `CGAL`.

As expected, every method has similar behavior for [int] and the respective [rat] cases.

## 4.5 Polynomials from geometric problems

The [3darr] polynomials have relatively big bitsize and allow us to observe the break even point of `BSDSC` and `RS`, see Figure 3. For small bitsizes the CF-Family methods perform better than the other methods, but `BSDSC` and `RS` outperform the CF-Family for bitsizes bigger than 1 200. `RS` is the most effective method for bitsizes over 600 bits. The behavior of the method is very good in this set, since it is fast enough for polynomials of bitsize lower than 600, and the fastest one for bitsizes over this threshold. However, the bitsize increase has less effect on `BSDSC` than on `RS` and the other methods. The low degree (in this case 12) is in favor of `BSDSC` for high bitsizes.

**Table 5.** [3darr] polynomials.

| bits | Sleeve | Symbnum | CF-Fam. | DSC | BSDSC | RS |
|---|---|---|---|---|---|---|
| [3darr] : Degree 12 | | | | | | |
| 200 | 4.26 | 4.10 | **3.44** | 5.48 | 5.82 | 4.58 |
| 400 | 7.46 | 7.36 | **4.30** | 6.42 | 5.92 | 4.96 |
| 600 | 10.78 | 10.74 | **5.36** | 7.28 | 6.26 | **5.22** |
| 800 | 14.34 | 14.38 | **6.48** | 8.36 | 6.90 | **5.72** |
| 1000 | 18.60 | 18.58 | **7.24** | 9.84 | 7.30 | **6.28** |
| 1200 | 22.98 | 22.92 | 7.70 | 10.64 | **7.58** | **6.64** |
| 1400 | 27.86 | 27.88 | 8.32 | 11.78 | **7.74** | **7.00** |
| 1600 | 33.78 | 34.12 | 9.06 | 13.38 | **7.82** | **7.30** |
| 1800 | 39.66 | 39.62 | 9.66 | 13.78 | **7.46** | **7.02** |
| 2000 | 46.54 | 46.58 | 9.94 | 15.24 | **7.68** | **7.50** |

**Figure 3.** [3darr] polynomials

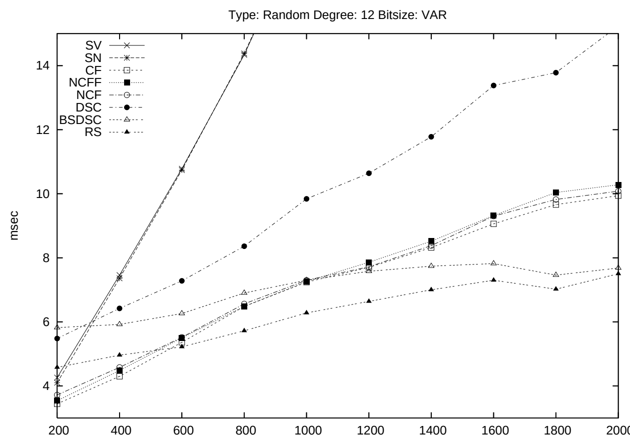



**Table 6.** [vorell] polynomials.

| Sleeve | Symbnum | CF | NCFF | NCF | DSC | BSDSC | RS |
|---|---|---|---|---|---|---|---|
| **195.2** | **194.2** | 394.4 | 787.4 | 265.6 | 759.2 | 1275.4 | 340.6 |

The [vorell] dataset contains polynomials having the highest degree of all the polynomials tested, namely degree 184. Moreover, the bitsize is considerably high (approx. 3 500 bits). The fastest methods in this case are `Sleeve` and `Symbnum`. Since the polynomials are random, approximate arithmetic suffices for real root isolation. `NCF` is very close to these methods. `NCFF` and `DSC` are equivalent, while `BSDSC` is not competitive. The degree of these polynomials slows down `BSDSC`, while `NCFF`'s factorization is slow due to the bitsize. `RS` is equivalent to `CF`, which are better than `NCFF` and `DSC`, but worse than `NCF` and the approximative methods.

# 5. CONCLUSION

We summarize the performance of various implementations of univariate solvers in several typical problem cases. Tables showing the method with the best performance for [rnd]/[bts] and [mgn] are presented in Figures 4 and 5. All three characteristics considered (degree, bitsize, root separation) are important in deciding which method is more effective for a specific problem.

From the general picture of the benchmarks, it is clear that there is no "best method" overall. Depending on the characteristics of the problem the performance of any method

**Figure 4.** Polynomials of random coefficients

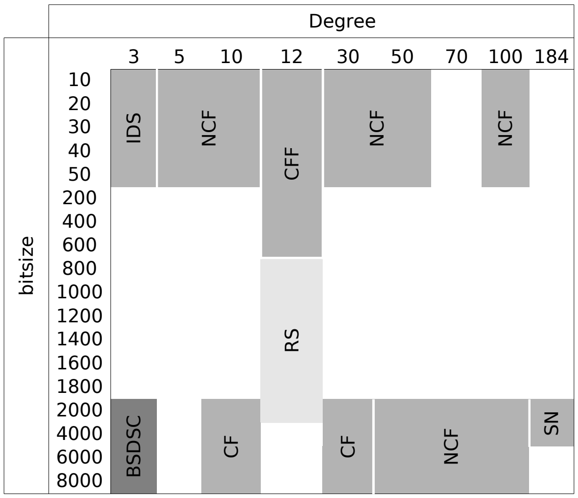
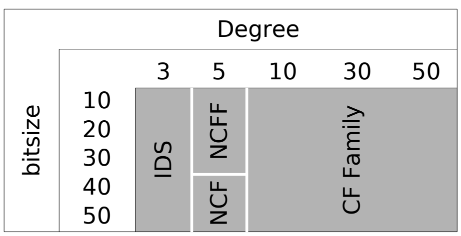



**Figure 5.** Polynomials with small real root separation




may change. However, one can say that for almost all instances the CF and the NCF solvers are among the best methods. The only exceptions are cases of polynomials having simultaneously low degree and big bitsize, as shown in Figure 4. In these cases BSDSC or RS show a better performance, where RS is slightly more effected by the bitsize than the BSDSC.

A remarkable observation is that the implementations of the theoretically exact methods are complete and that they always provided correct results throughout this extensive benchmarking procedure. In some cases though, RS, DSC or BSDSC may need too much time to finish. Nevertheless, given the time, they would finish by isolating all real roots. The implementations are numerically and combinatorially robust, which means that they comply with their theoretic exactness.

## 5.1 Summary for methods

We summarize our results with respect to the different approaches.

### *5.1.1 IDS*

Isolation and Discrimination Systems are applicable for polynomials with degree $\leq 4$ and were used within CF, Sleeve, Symbnum and Sturm. Their constant arithmetic complexity is consistent with the benchmark results. In particular, this is the case for polynomials with a small root separation, see the [mgn] instance in Table 1. However, the current implementation just applies exact arithmetic and is not adaptive at all. Hence, there is a clear disadvantage for input polynomials with moderate to large bitsize.

### *5.1.2 Sleeve and Symbnum*

The implementation of both solvers must be considered as preliminary, since Symbnum failed to isolate all roots in some cases. In general it was not possible to observe a difference among the two solvers, which leads to the conjecture that the runtime of Symbnum is clearly dominated by the time spent within Sleeve.

The Sleeve method is very effective in small problems, where an approximation is able to isolate the roots correctly. Even for [mgn], which is the most difficult case for approximation methods. However, when it comes to higher degrees and bitsizes, it is either much slower than the other methods or fails to isolate all the real roots correctly.

For polynomials of random coefficients and degree 100 and bitsize up to 50, both methods are close to NCF and faster than all other methods. The randomness of the coefficients is in favor of these approximation methods. However, in the case of polynomials derived from geometric problems, we observe different behavior depending on the problem. In the low degree polynomials in [3darr] the approximation methods are sensitive to the bitsize increase and thus not competitive to the exact methods. The same holds for the [bts] datasets. This was not expected since both datasets contain rather easy polynomials. On the other hand, both solvers perform very well in isolating the real roots of the degree 184 resultant that appears in Voronoi diagram of ellipses problems. The approximation methods correctly isolate the real roots considerably faster than the fastest exact method, namely NCF. This can be explained by the number (8) of well separated real roots with respect to the high degree (184) of the polynomials within this dataset.

Thus, the fact that the coefficients of a polynomial are random is not sufficient for the approximation methods to outperform the exact ones.

The performance of real solvers on "easy" polynomials (few and well separated real roots) can be improved, when a numeric filter is enough to (almost) separate the roots, provided that the conversion of the input is not costly.

However, on difficult polynomials, where the total time spent is essentially due to (the depth of the subdivision tree depending on) the subdivision strategy, the symbolic-numeric method is considerably slower than other methods and/or do not provide the correct result.

### *5.1.3 Sturm*

As expected the Sturm solver is always among the slowest methods. This is the case for polynomials with a small root separation as well as for polynomials with large coefficients. There are two reasons for this behavior. First, the algorithm needs a big amount of memory, since it computes a polynomial remainder sequence, the size of which is quadratic with respect to the degree of the polynomials. Second, it performs, from the beginning and almost always, computations with numbers of bitsize equal to the theoretical separation bound, which is quite time consuming. Nevertheless, the algorithm is of great theoretical importance, while its straightforward implementation makes it attractive for very small examples and educational purposes.

#### 5.1.4 DSC

Though the `DSC` solver was also only incorporated for reference, it performed considerably better than `Sturm`. For the `[mgn]` instances as well as for `[bts]` instances up to degree 30 it was even faster than `RS`. And for several instances with small degree and moderate bitsize the times were similar or even better than other methods. However, for polynomials with high degree or large coefficients the `DSC` solver is not competitive.

#### 5.1.5 CF-Family

Even for moderate degrees, only the methods based on Continued Fractions are capable of solving Mignotte polynomials, since there is a rational number with a simple continued fraction expansion between the real roots that are very close together. The efficiency of the method is due to its nature, that does not depend on the separation of the roots, but rather on their "distribution" on the real axis. This makes the methods based on Continued Fractions, the only ones competitive on polynomials with small real root separation.

Overall, one can say that the CF-Family is the method of choice for moderate to large degrees and for small or moderate bitsizes. However, due to the use of exact arithmetic throughout the algorithm the solvers are expected to be outperformed by `RS` and `BSDSC` for easy polynomials with large bitsize and moderate degree, see Figure 1 and Figure 3.

The `NCFF` solver was often much slower than the other solvers, which was caused by the additional time spent within the factorization of the polynomials. That is, it is only advisable to use this method in case the polynomials are known (or likely) to factorize and in case this gives some advantage in subsequent steps. For instance, the method was among the slowest for the `[vorell]` dataset, since it tried to factorize a polynomial of relative high degree, 184, with big bitsize, that turn out to be irreducible.

Our suggestion is to decouple such a step from the actual solver and should be applied on a case by case basis.

#### 5.1.6 BSDSC

The `BSDSC` is the best solver for easy polynomials of low degree and large coefficients. The main disadvantage is that the number of bits which is required to certify the roots considerably depends on the degree of the polynomials. Therefore, it is often much slower than the `CF`-Family and sometimes also slower than the `RS` solver. However, it seems almost independent from the bitsize of the input polynomials, see Figure 1 and Figure 3.

Another advantage of the `BSDSC` is that it is able to handle coefficient types other than integers (rationals). In particular the `BSDSC` is intended to work over algebraic extensions, as it is reported in [19, 10, 11], for example, it can isolate roots of $f(x, y)_{|x=\alpha}$, where $\alpha$ is an algebraic number. This is the actual strength of the `BSDSC`. Currently is the only method recommended for non rational coefficient types.

#### 5.1.7 RS

In general the behavior of the `RS` solver is similar to the behavior of the `BSDSC` solver. However, for many instances it is faster than the `BSDSC` due to the fact that it is less effected by the degree of the polynomials. On the other hand it is more effected by the bitsize. A serious exception are the Mignotte polynomials, for which `RS-solver` is considerably slow. The problem is due to the high memory consumption of the algorithm for these instances. For degrees higher than 30 the method would run out of memory, since it is forced to go very deep to the subdivision tree to isolate the roots.

## 6. FURTHER WORK

Our current experiments with polynomials with only integer and/or rational roots are not conclusive for the behavior of the tested implementations. More experiments are needed, with polynomials of high degree and big bitsize.

It should be possible to improve the performance and robustness of the symbolic-numeric techniques which can serve as a filter for all exact methods presented here.

Isolation and Discrimination Systems (IDS) could be further improved by exploiting certified approximate techniques, such as multi-precision floating point interval arithmetic. This would lead to a more adaptive behavior.

Given the advantages of the bitstream method for huge bit sizes (and other coefficient types) as well as the advantages of the methods based on Continued Fractions for small separation bound, it seems worthwhile exploiting the bitstream idea for methods based on Continued Fractions.

## Acknowledgments

All authors acknowledge partial support by IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-006413-2 (ACS - Algorithms for Complex Shapes). I. Emiris and B. Mourrain are partially supported by Marie-Curie Network "SAGA", FP7 contract PITN-GA-2008-214584. B. Mourrain and E. Tsigaridas acknowledge partial support by contract ANR-06-BLAN-0074 "Decotes". Z. Zafeirakopoulos is supported by the Austrian Science Fund (FWF), under project no.11 in the DK on Computational Mathematics. The authors are grateful to Luis Peñaranda for his help with the `LORIA` kernel and to Sebastian Limbach for his help with the `MPI` kernel.

## References

[1] A. Akritas, A. Strzebonski, and P. Vigklas. Improving the performance of the continued fractions method using new bounds of positive roots. *Nonlinear Analysis*, 13(3):265–279, 2008.

[2] M. H. Austern. *Generic Programming and the STL*. Addison-Wesley, 1998.

[3] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, L. Kettner, K. Mehlhorn, J. Reichel, S. Schmitt, E. Schömer, and N. Wolpert. Exacus: Efficient and exact algorithms for curves and surfaces. In *13th Annual European Symp. on Algorithms (ESA)*, volume 3669 of *LNCS*, pages 155–166, Palma de Mallorca, Spain, 2005. Springer.

[4] E. Berberich, M. Hemmer, M. I. Karavelas, and M. Teillaud. Revision of the interface specification of algebraic kernel. Technical Report ACS-TR-243301-01, INRIA, MPI and NUA, 2007.

[5] D. Bini and G. Fiorentino. Design, analysis, and implementation of a multiprecision polynomial rootfinder. *Numerical Algorithms*, pages 127–173, 2000.

[6] G. Collins, J. Johnson, and W. Krandick. Interval arithmetic in cylindrical algebraic decomposition. *J. of Symbolic Computation*, 34:143–155, 2002.

[7] G. E. Collins and A. G. Akritas. Polynomial real root isolation using descarte's rule of signs. In *Proc. $3^{rd}$ ACM Symp. on Symbolic and Algebraic Comp.*, pages 272–275, NY, USA, 1976.

[8] J. H. Davenport. Cylindrical algebraic decomposition. Technical Report 88–10, School of Mathematical Sciences, University of Bath, England, 1988.

[9] Z. Du, V. Sharma, and C. K. Yap. Amortized bound for root isolation via Sturm sequences. In D. Wang and L. Zhi, editors, *Int. Workshop on Symbolic Numeric Computing*, pages 113–129, School of Science, Beihang University, Beijing, China, 2005. Birkhauser.

[10] A. Eigenwillig. *Real Root Isolation for Exact and Approximate Polynomials Using Descartes' Rule of Signs*. PhD thesis, Universität des Saarlandes, Saarbrücken, 2008.

[11] A. Eigenwillig, L. Kettner, W. Krandick, K. Mehlhorn, S. Schmitt, and N. Wolpert. A descartes algorithm for polynomials with bit-stream coefficients. In *Proc. 8th Int. Workshop on Computer Algebra in Scient. Comput. (CASC)*, volume 3718 of *LNCS*, pages 138–149, Kalamata, Greece, 2005. Springer.

[12] A. Eigenwillig, V. Sharma, and C. K. Yap. Almost tight recursion tree bounds for the Descartes method. In *Proc. Annual ACM ISSAC*, pages 71–78, New York, USA, 2006.

[13] I. Emiris, E. Tsigaridas, and G. Tzoumas. Predicates for the exact Voronoi diagram of ellipses under the Euclidean metric. *Intern. J. Comp. Geometry & Appl.*, 18(6):567–597, 2008. Special Issue on SoCG'06.

[14] I. Emiris, E. Tsigaridas, and G. Tzoumas. Voronoi diagram of ellipses in CGAL. In *Proc. Europ. Works. Comp. Geom.*, pages 87–90, Nancy, France, 2008.

[15] I. Emiris and G. Tzoumas. Exact and efficient decision of the InCircle predicate for parametric ellipses and smooth convex objects. *Computer-Aided Design (Elsevier)*, 40:691–700, 2008. Special issue on SPM'07.

[16] I. Z. Emiris, B. Mourrain, and E. P. Tsigaridas. Real Algebraic Numbers: Complexity Analysis and Experimentation. In *Reliable Implementations of Real Number Algorithms: Theory and Practice*, volume 5045 of *LNCS*, pages 57–82. Springer Verlag, 2008.

[17] I. Z. Emiris and E. P. Tsigaridas. Real algebraic numbers and polynomial systems of small degree. *Theoretical Computer Science*, 409(2):186 – 199, 2008.

[18] M. Hemmer and D. Hülse. Generic implementation of a modular gcd over algebraic extension fields. In *25th European Workshop on Computational Geometry*, pages 321–324, Brussels, Belgium, 2009. Université Libre de Bruxelles.

[19] M. Hemmer and S. Limbach. Benchmakrs on a generic univariate algebraic kernel. Technical Report ACS-TR-243306-03, Algorithms for Complex Shapes with certified topology and numerics, Max Planck Institut für Informatik, Saarbrücken, Germany, 2007.

[20] J. R. Johnson, W. Krandick, K. Lynch, D. Richardson, and A. Ruslanov. High-performance implementations of the Descartes method. In *Proc. Annual ACM ISSAC*, pages 154–161, NY, 2006.

[21] M. I. Karavelas. Preliminary cross-benchmarks of the MPI and NUA univariate algebraic kernels. Technical Report ACS-TR-243306-05, National University of Athens, 2008.

[22] W. Krandick. Isolierung reeller nullstellen von polynomen. In J. Herzberger, editor, *Wissenschaftliches Rechnen*, pages 105–154. Akademie-Verlag, Berlin, 1995.

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

[24] S. Lazard, L. Peñaranda, and E. Tsigaridas. Univariate algebraic kernel and application to arrangements. In *Symp. of Experimental Algorithms (SEA)*, 2009. (to appear).

[25] M. Mignotte. *Mathematics for computer algebra*. Springer-Verlag, New York, 1991.

[26] B. Mourrain, P. Pavone, P. Trébuchet, E. P. Tsigaridas, and J. Wintz. SYNAPS, a library for dedicated applications in symbolic numeric computations. In M. Stillman, N. Takayama, and J. Verschelde, editors, *IMA Vol. in Math. and its Applications*, pages 81–110. Springer, 2007.

[27] B. Mourrain, M. Vrahatis, and J. Yakoubsohn. On the complexity of isolating real roots and computing with certainty the topological degree. *J. Complexity*, 18(2), 2002.

[28] V. Pan. Univariate polynomials: Nearly optimal algorithms for numerical factorization and rootfinding. *J. Symbolic Computation*, 33(5):701–733, 2002.

[29] F. Rouillier and P. Zimmermann. Efficient isolation of polynomial real roots. *J. of Computational and Applied Mathematics*, 162(1):33–50, 2003.

[30] V. Sharma. Complexity of real root isolation using continued fractions. *Theor. Comput. Sci.*, 409(2):292–310, 2008.

[31] E. P. Tsigaridas. *Algebraic algorithms and applications to geometry*. PhD thesis, National Kapodistrian University of Athens, 2006.

[32] E. P. Tsigaridas and I. Z. Emiris. On the complexity of real root isolation using continued fractions. *Theor. Comput. Sci.*, 392(1-3):158–173, 2008.