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 5000 polynomials from many different settings, which have maximum coefficient bitsize up to bits 8000, 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

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, 27, 8, 29, 16, 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 \(O(d^3\tau^2)\), 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 \(O_B(d^3\tau)\), which can be further improved to \(O_B(d^2\tau)\), 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, 21, 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 [25, 11, 3] 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 com-
pare 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\(^2\). 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\(^2\) 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 bit-size, and random polynomials of degree 100, and (ii) random polynomials with fixed bitsize (32 and 100) and varying degree (up to 2000), 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-numerical 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 5000 polynomials from many different settings, which have maximum coefficient bitsize up to bits 8000, 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\(^3\) 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\(^3\).

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\(^5\) we propose possible directions for future work in univariate real solving.

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 \(\mathcal{O}_B\)-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\(^4\) 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\(^{25}\) for more details.

The library provides 4 exact methods for real root isolation of univariate polynomial with integer coefficients, namely Sturm, CF, NCF 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 \(O(1)\), whereas the bit-complexity is \(O_B(\tau)\), see\(^7\)\(^8\)\(^1\)\(^1\)\(^2\)\(^4\) 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
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 $O_B(d^4 + d^4 \tau^2)$, see also [10, 11, 1] 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 [11]. We refer the reader to [52] [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 $O_B(d^4 \tau^2)$ [32], while the average case complexity is $O_B(d^4 + d^4 \tau^2)$ [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 re-alroot of mathemagix. In case the polynomial degree is ≤ 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 [13]. 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 ≤ 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 ≤ 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 [10].

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 [13] [10] [11], for example, it can isolate roots of $f(x, y)_{\alpha = \alpha}$, 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 [13] 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 $O_B(d^4 \tau^2)$. 
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 BSDC. According to [10], the worst case complexity of BSDSC is $\tilde{O}_d(d^5 \tau^2)$.

### The VEGAS Kernel

The kernel which is developed by the VEGAS group at LORIA is based on the RS library\footnote{http://fgbbrs.lip6.fr} which in turn is developed by the SALSA group at INRIA-Rocquencourt, see [24, 25]. 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.

#### RS

The RS solver is based on the interval Descartes algorithm [3]. 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.

### 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 [10]. 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.

- **[mgn]** datasets contain Mignotte polynomials [20], 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.

### 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\footnote{http://gmplib.org/} (version 4.2) and NTL\footnote{http://www.shoup.net/ntl} (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
from NTL integer types. Time, in msec, was measured using the clock() function of the time 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 [3darrl] and [mgnl] datasets the polynomials 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 which originate from the various datasets. The best times in each row are emphasized. Note that the table does not list

<table>
<thead>
<tr>
<th>Method</th>
<th>mgnl</th>
<th>[int]</th>
<th>[rat]</th>
<th>[bts]</th>
<th>[mgnl]</th>
<th>[int]</th>
<th>[rat]</th>
<th>[bts]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.288</td>
<td>0.282</td>
<td>0.264</td>
<td>0.268</td>
<td>0.279</td>
<td>0.272</td>
<td>0.262</td>
<td>0.236</td>
</tr>
<tr>
<td>20</td>
<td>0.270</td>
<td>0.256</td>
<td>0.256</td>
<td>0.260</td>
<td>0.254</td>
<td>0.248</td>
<td>0.242</td>
<td>0.230</td>
</tr>
<tr>
<td>30</td>
<td>0.262</td>
<td>0.258</td>
<td>0.258</td>
<td>0.262</td>
<td>0.256</td>
<td>0.250</td>
<td>0.244</td>
<td>0.236</td>
</tr>
<tr>
<td>40</td>
<td>0.252</td>
<td>0.248</td>
<td>0.248</td>
<td>0.252</td>
<td>0.246</td>
<td>0.240</td>
<td>0.234</td>
<td>0.226</td>
</tr>
<tr>
<td>50</td>
<td>0.230</td>
<td>0.224</td>
<td>0.224</td>
<td>0.228</td>
<td>0.222</td>
<td>0.216</td>
<td>0.210</td>
<td>0.202</td>
</tr>
<tr>
<td>60</td>
<td>0.210</td>
<td>0.204</td>
<td>0.204</td>
<td>0.208</td>
<td>0.202</td>
<td>0.196</td>
<td>0.190</td>
<td>0.182</td>
</tr>
<tr>
<td>70</td>
<td>0.190</td>
<td>0.184</td>
<td>0.184</td>
<td>0.188</td>
<td>0.182</td>
<td>0.176</td>
<td>0.170</td>
<td>0.162</td>
</tr>
<tr>
<td>80</td>
<td>0.170</td>
<td>0.164</td>
<td>0.164</td>
<td>0.168</td>
<td>0.162</td>
<td>0.156</td>
<td>0.150</td>
<td>0.142</td>
</tr>
<tr>
<td>90</td>
<td>0.150</td>
<td>0.144</td>
<td>0.144</td>
<td>0.148</td>
<td>0.142</td>
<td>0.136</td>
<td>0.130</td>
<td>0.122</td>
</tr>
<tr>
<td>100</td>
<td>0.130</td>
<td>0.124</td>
<td>0.124</td>
<td>0.128</td>
<td>0.122</td>
<td>0.116</td>
<td>0.110</td>
<td>0.102</td>
</tr>
</tbody>
</table>

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 [3darrl] and [mgnl] datasets the polynomials 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.

### 4.2 Polynomials of small real root separation

For the [mgnl] 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 [mgnl] 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 [mgnl] instances known to be square free, which was indicated by a flag. Thus, the implementations for CF, NCF and NCF can be considered as identical since they only differ in the way the square free factorization is implemented.

For the [mgnl] 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 [mgnl] 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 itself. This also explains the difference in the CF-Family. Table 3 shows the results.
Table 2. [sgx] polynomials.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Sturm</th>
<th>CF Fam.</th>
<th>DSC</th>
<th>BSDSC</th>
<th>RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>[sgx]: Degree 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3.2</td>
<td>1.6</td>
<td>1.8</td>
<td>2.0</td>
<td>6.8</td>
</tr>
<tr>
<td>20</td>
<td>7.6</td>
<td>2.0</td>
<td>2.0</td>
<td>2.2</td>
<td>15.2</td>
</tr>
<tr>
<td>30</td>
<td>10.8</td>
<td>3.0</td>
<td>4.0</td>
<td>3.6</td>
<td>26.0</td>
</tr>
<tr>
<td>40</td>
<td>15.8</td>
<td>3.6</td>
<td>4.2</td>
<td>4.2</td>
<td>40.8</td>
</tr>
<tr>
<td>50</td>
<td>21.4</td>
<td>5.0</td>
<td>6.0</td>
<td>4.4</td>
<td>57.2</td>
</tr>
<tr>
<td>[sgx]: Degree 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>7.6</td>
<td>1.8</td>
<td>4.0</td>
<td>4.0</td>
<td>11.8</td>
</tr>
<tr>
<td>20</td>
<td>19.6</td>
<td>3.6</td>
<td>7.8</td>
<td>8.0</td>
<td>26.4</td>
</tr>
<tr>
<td>30</td>
<td>33.4</td>
<td>4.8</td>
<td>12.4</td>
<td>12.0</td>
<td>43.6</td>
</tr>
<tr>
<td>40</td>
<td>52.4</td>
<td>7.6</td>
<td>17.6</td>
<td>15.8</td>
<td>68.0</td>
</tr>
<tr>
<td>50</td>
<td>77.6</td>
<td>9.4</td>
<td>24.6</td>
<td>19.8</td>
<td>95.2</td>
</tr>
</tbody>
</table>

Table 3. [bts] polynomials.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Symbnum</th>
<th>CF</th>
<th>NCFF</th>
<th>NCF</th>
<th>DSC</th>
<th>BSDSC</th>
<th>RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>[bts]: Degree 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>2.0</td>
<td>2.2</td>
<td>1.1</td>
<td>1.9</td>
<td>1.9</td>
<td>2.2</td>
<td>2.7</td>
</tr>
<tr>
<td>400</td>
<td>7.10</td>
<td>7.09</td>
<td>7.17</td>
<td>2.11</td>
<td>1.40</td>
<td>1.42</td>
<td>1.56</td>
</tr>
<tr>
<td>800</td>
<td>7.82</td>
<td>7.85</td>
<td>7.86</td>
<td>4.80</td>
<td>2.48</td>
<td>1.95</td>
<td>1.68</td>
</tr>
<tr>
<td>[bts]: Degree 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>2.1</td>
<td>2.0</td>
<td>1.4</td>
<td>1.3</td>
<td>1.7</td>
<td>3.0</td>
<td>4.0</td>
</tr>
<tr>
<td>4000</td>
<td>7.72</td>
<td>7.64</td>
<td>7.57</td>
<td>3.84</td>
<td>2.01</td>
<td>1.94</td>
<td>2.11</td>
</tr>
<tr>
<td>8000</td>
<td>7.16</td>
<td>7.16</td>
<td>7.15</td>
<td>5.92</td>
<td>2.84</td>
<td>2.40</td>
<td>2.16</td>
</tr>
<tr>
<td>[bts]: Degree 20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td>3.56</td>
<td>3.52</td>
<td>3.34</td>
<td>3.34</td>
<td>3.01</td>
<td>3.52</td>
<td>3.75</td>
</tr>
<tr>
<td>40000</td>
<td>7.72</td>
<td>7.64</td>
<td>7.57</td>
<td>3.84</td>
<td>2.01</td>
<td>1.94</td>
<td>2.11</td>
</tr>
<tr>
<td>80000</td>
<td>7.16</td>
<td>7.16</td>
<td>7.15</td>
<td>5.92</td>
<td>2.84</td>
<td>2.40</td>
<td>2.16</td>
</tr>
</tbody>
</table>

Figure 1. [bts] polynomials of degree 50

Figure 2. [bts] polynomials of degree 100

Figure 3. [bts] polynomials of degree 500

For polynomials of small degree the Descartes solvers are remarkably fast, in particular they are even faster than CF, which uses IDS for degrees ≤ 4. For moderate degrees the CF method is faster, whereas the NCF method is the fastest for degrees ≥ 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 CF method is the slowest method due to the extra time spent in the, in this case, useless factorization.

Besides NCF, 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 4 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. Sleevel 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 8000 bits, we can not observe the break even point for BSDSC due to the huge constant overhead. As one can see in Table 5 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

The [int] and [rat] instances indeed contain polynomi-
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 affected by the time spent, not by root isolation, but within square-free factorization (if this operation is available). 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 2. 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.

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] and [bts] are presented in Figures 1 and 2. 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

<table>
<thead>
<tr>
<th>Bits</th>
<th>Sleeve</th>
<th>Symbnum</th>
<th>CF-Fam.</th>
<th>DSC</th>
<th>BSDSC</th>
<th>RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>6.46</td>
<td>6.48</td>
<td>6.32</td>
<td>6.26</td>
<td>6.23</td>
<td>6.20</td>
</tr>
<tr>
<td>800</td>
<td>11.06</td>
<td>11.06</td>
<td>11.00</td>
<td>11.00</td>
<td>11.00</td>
<td>11.00</td>
</tr>
</tbody>
</table>

Figure 3. [3darr] polynomials

Type: Random Degree: 12 Bitsize: VAR

Table 7. [voroll] polynomials.

The [voroll] 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. NCFF 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 NCFF 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] and [bts] are presented in Figures 1 and 2. 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
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 \([\text{mgn}]\) 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 \([\text{mgn}]\), 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 \([\text{3darr}]\) the approximation methods are sensitive to the bitsize increase and thus not competitive to the exact methods. The same holds for the \([\text{bts}]\) 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 [mgp] 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 2.

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 [10] [14] [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


