DOI: 10.15476/ELTE.2021.151

Eötvös Loránd University
Faculty of Informatics
Department of Computer Algebra

Algorithms and computations related to algebraic numbers

Theses of the PhD dissertation

Uray Marcell János

Supervisor:
Burcsi Péter
(associate professor)

Doctoral school:	Doctoral School of Informatics
Head of doctoral school:	Dr. Csuhaj Varjú Erzsébet (professor)
Doctoral program:	Numeric and Symbolic Computations
Head of program:	Dr. Weisz Ferenc (professor)

Budapest, 2021

The dissertation is about algorithms of algebraic numbers and related theoretical questions. An *algebraic number* is a number that is a root of an integer polynomial, e.g. $\sqrt{2}$ is the root of $x^2 - 2$.

An important aspect of the discussed algorithms is that they calculate *exactly*, not *numerically* (i.e. with approximations). So $\sqrt{2}$ is never 1.4142, but always exactly the $\sqrt{2}$ mathematical object. In this way we avoid the usual problem of numerical calculations, the rounding errors, but we face other kind of problems instead: the exact algorithms are usually slower than the numerical ones, and they often suffer from *coefficient explosion*, which means that the integers needed for the exact representation grow too fast, often exponentially fast during the algorithm, which affects the running time seriously. Avoiding this problem and maintaining the running time (i.e. ensuring polynomial time) is an important aspect of exact algorithms.

1. Thesis: Operations in algebraic number fields. *We analyse the computational costs of many operations in algebraic number fields using exact arithmetic: we calculate how much the representing coefficients can grow during the operations, and give bounds on their running time.*

Exact algorithms with algebraic numbers are important, and they are described in many papers and books (e.g. [6, 7]). However, the computational costs of these algorithms are often not obvious to calculate, because the bit complexity depends on how much the representing multi-precision integers grow during the computation. There are some earlier results on this topic, e.g. [8], but they are not general enough. We use some results from these, but otherwise, up to our knowledge, such general results that are presented in the dissertation have not been calculated by others, and some special operators that we consider have not been considered by others at all.

Algebraic numbers can be represented efficiently in *algebraic number field*s: fix an algebraic number θ, and consider all numbers that can be obtained from it by the four basic operations. They can be represented in the form $a_0 + a_1\theta + a_2\theta^2 + \ldots + a_{m-1}\theta^{m-1}$, where a_i's are rational numbers, and m is the degree of θ. In many cases we can clear off the denominators of the rational coefficients, so we assume that they are already integers.

We introduce the notation $\mathrm{s}(\alpha)$ for the maximal length of the integer coefficients of α, i.e., using the notation above: $\mathrm{s}(\alpha) := \log \max_i |a_i|$. Then it is easy to see that for example $\mathrm{s}(\alpha + \beta) \leq \max(\mathrm{s}(\alpha), \mathrm{s}(\beta)) + \log 2$. It is less obvious, but we also show that for example $\mathrm{s}(\alpha\beta) \leq \mathrm{s}(\alpha) + \mathrm{s}(\beta) + C$, where C depends on the number field (we give it explicitly in the dissertation). We give similar results for all four basic operations, and also calculate a lower and upper bound for $|\alpha|$ in terms of $\mathrm{s}(\alpha)$.

Furthermore, we examine the running time of the main operations in algebraic number fields. To be as general as possible, we introduce the following notation: let $\mathrm{Mul}(A)$ be the number of machine operations needed to multiply two integers of length at most A. For the standard multiplication algorithm, $\mathrm{Mul}(A) = O(A^2)$, but for the Schönhage–Strassen fast multiplication, which is faster for large integers, $\mathrm{Mul}(A) = O(A \log A \log\log A)$. We give all our results in terms of $\mathrm{Mul}(\cdot)$. It is easy to see for example that multiplying an algebraic number α by a B-bit integer requires at most $m\,\mathrm{Mul}(\mathrm{s}(\alpha), B)$ time. We calculate similar formulas for the less obvious operations, including multiplication and division of two algebraic numbers.

We go further, and examine some operations that are special to real algebraic numbers. One of them is comparison (e.g. $\alpha < \beta$). In most cases this is trivial to perform using approximation, but if the two numbers are very close to each other, then even the most precise floating-point number type of the computer may fail to notice the difference. We examine a possible way to decide the question exactly, and we calculate its worst-case complexity. We do the same for another operation, the integer rounding. These results are heavily used in the next section.

2. Thesis: LLL algorithm in algebriac number fields. *We apply the LLL lattice reduction algorithm to algebraic number fields, and prove that it can be performed in polynomial time even with exact arithmetic.*

The LLL algorithm is a lattice basis reduction algorithm, which converts a basis of a lattice to a reduced basis of the same lattice. It was developed by Lenstra, Lenstra and Lovász in [9], and they analysed its running time if the input vectors are in $\mathbb{Z}^n$. There are also many efficient generalisations for $\mathbb{R}^n$ using floating-point arithmetic. But we consider another generalisation: we analyse the LLL algorithm on vectors over a real algebraic number field using exact arithmetic.

Exact calculation in the LLL algorithm might not seem very useful, since for many practical applications, the goal is not to get the exact result, but to find a well-reduced basis or a sufficiently short vector, and the floating-point LLL versions are much faster. However, we still think that the analysis of the exact algorithm deserves interest, first because it is interesting from a theoretical point of view, but also there are applications where the exact values in the reduction are needed, for example in [10].

The algorithm is structured around a main while-loop, and modifies the vectors step by step. By looking at its structure, it is not obvious how many iterations it performs, or even if it terminates at all. The authors of [9] proved that it always terminates, and bounded its running time in $\mathbb{Z}^n$. In the dissertation, we generalise this calculation for algebraic integers. This is not an easy task, since [9] used the property of $\mathbb{Z}$ that all positive numbers are at least 1, which is not true for algebraic integers, as they can be arbitrarily close to 0.

We analyse the LLL algorithm in algebraic number fields, and give an upper bound on its running time. This is calculated in three steps: first we give an upper bound on the number of iterations in the main while-loop, then we calculate bounds on the size of the variables during the algorithm using our $\mathrm{s}(\cdot)$ notation, then we calculate the running time of one iteration of the main while-loop. We combine these three results and get the running time of whole algorithm, which turns out to be polynomial. We use our formulas for the costs of the operations from the previous part, and give the result generally using the $\mathrm{Mul}(\cdot)$ notation.

3. Thesis: Characterisation of expansive polynomials. *We give an exact and efficient method to decide whether a polynomial is expansive. The algorithm does not suffer from coefficient explosion, and it is guaranteed to run in polynomial time.*

We examine special algebraic numbers which are greater than 1 in absolute value, and so are all the other roots of the defining polynomial (whose root the examined number is). On the complex plane, this means that all roots of the polynomial are outside of the unit circle. Such polynomials are called *expansive polinomial*s.

The question is how we can decide whether a polynomial with integer coefficients is expansive (without calculating its roots). There are some methods in the literature, for example the Schur–Cohn test [11], which is the key component of the Lehmer–Schur algorithm for complex root finding. The Schur–Cohn test generates a sequence of polynomials of decreasing degree using a simple rule, and examines a simple condition on each polynomial, namely that its constant term is greater than its leading coefficient in absolute value. If this holds for each one, then the original polynomial is expansive. For example, the polynomial $2x^4 + x^3 - 14x^2 - 4x + 24$ generates the following sequence:

$$\begin{aligned}
2x^4 + x^3 - 14x^2 - 4x + 24,\\
32x^3 - 308x^2 - 98x + 572,\\
-173040x^2 - 46200x + 326160,\\
-23063040000x + 76437504000,\\
5310788203708416000000.
\end{aligned}$$

We can see that the test passes, so the polynomial is expansive.

The example also shows something else: that the Schur–Cohn test suffers from coefficient explosion. Its recursive rule allows the coefficients to double their length in each step, which may lead to exponential coefficient growth. The algorithm is often calculated with numerical approximations, where this is not a problem, but then the larger numbers may not be represented precisely, and so the result is not guaranteed to be correct.

We introduce a new exact method to decide expansivity, which avoids coefficient explosion. The method constructs determinants from the coefficients of the polynomial using a simple rule. For example for degree 4, if $f(x) = ax^4 + bx^3 + cx^2 + dx + e$, these determinants are the following:

$$\begin{vmatrix} e-a \end{vmatrix} \qquad \begin{vmatrix} e-b & d-a \\ -a & e \end{vmatrix} \qquad \begin{vmatrix} e-c & d-b & c-a \\ -b & e-a & d \\ -a & & e \end{vmatrix}$$

$$\begin{vmatrix} e+a \end{vmatrix} \qquad \begin{vmatrix} e+b & d+a \\ a & e \end{vmatrix} \qquad \begin{vmatrix} e+c & d+b & c+a \\ b & e+a & d \\ a & & e \end{vmatrix}$$

According to our method, the polynomial is expansive if and only if all these determinants are positive (and also some trivial conditions hold). Continuing the earlier example, if $f(x) = 2x^4 + x^3 - 14x^2 - 4x + 24$, then these determinants are 22, 540, 19200, 26, 604 and 6960 respectively, and since they are all positive, it confirms that the polynomial is expansive. We can see that these numbers are much smaller than those that the Schur–Cohn test produces.

In the dissertation we prove the validity of our method in a general form, using the Schur–Cohn test. We also calculate the worst-case complexity of the algorithm, proving that it runs in polynomial time.

4. Thesis: Lower bounds on the expansivity gap. *We examine how close the roots of an expansive integer polynomial can be to the unit circle, and give lower bounds on this distance in terms of the degree and the size of the coefficients.*

The *expansivity gap* of an expansive polynomial is the distance between the root with the smallest size and the unit circle. We examine how small this quantity can be if the coefficients are integers.

Without further restrictions, this can be arbitrarily small, for example the expansivity gap of the linear polynomial $Ax-(A+1)$ is $1/A$, whose limit is zero if $A \to \infty$. Therefore we ask the question in this way: how small can it be if the degree is fixed and the coefficients have a fixed bound?

We give multiple answers to the question depending on how the coefficients are measured. We can bound the *height* of the polynomial, i.e. the largest absolute value of the coefficients, or its *length*, i.e. the sum of the absolute value of the coefficients, or we can restrict only one or two special coefficients like the constant term and the leading coefficient. We consider four cases, and calculate an explicit lower bound on the expansivity gap in each case in terms of the degree and the chosen measure of the coefficients.

We use several tools for the calculations. One of them is Liouville's inequality, which helps to bound distances between algebraic numbers. Using this, we prove for example that the expansivity gap of a polynomial of degree n and constant term a_0 is at least $1/(2^{\binom{n}{2}}|a_0|^{n-1}+1)$. Another tool is given by the determinants presented earlier, which we use to derive more complicated lower bounds. All of these bounds have in common that the quantity measuring the size of the coefficients (e.g. $|a_0|$ in the example above) has the exponent $n-1$. They are multiplied however by different n-dependent factors (e.g. $2^{\binom{n}{2}}$ in the example above, and this is the largest one of the four).

At the end of the dissertation we give an application of the calculated bounds: we can use them to estimate the computational complexity of certain algorithms related to generalised number systems.

5. Thesis: Barely expansive polynomials. *We construct a family of expansive integer polynomials with roots very close to the unit circle. This shows that our lower bounds on the expansivity gap are almost sharp.*

Considering the bounds mentioned above, the question naturally arises whether they can be improved or they are sharp. Proving sharpness is achieved by presenting polynomials whose expansivity gap has exactly the same order as these bounds.

For each degree n and each sufficiently large integer A we define a certain polynomial, and estimate its expansivity gap. We find that it is approximately $1/2A^{n-1}$, which is similar to our bounds, except that the latters have a greater, n-dependent factor instead of 2. Nevertheless, this shows that for fixed n, the bounds are sharp.

Our defined polynomials for the first few n are as follows:

$$\begin{aligned}
n=2: &\quad (A-1)x^2+(A-1)x+A,\\
n=3: &\quad -x^3+(A-1)x^2+(A-2)x+A,\\
n=4: &\quad -x^4-2x^3+(A-2)x^2+(A-2)x+A,\\
n=5: &\quad -x^5-3x^4-5x^3+(A-4)x^2+(A-3)x+A,\\
n=6: &\quad -x^6-4x^5-9x^4-12x^3+(A-9)x^2+(A-5)x+A,\\
n=7: &\quad -x^7-5x^6-14x^5-25x^4-30x^3+(A-21)x^2+(A-10)x+A.
\end{aligned}$$

In the dissertation we prove that these polynomials are indeed expansive (for sufficiently large A). For this, we use the determinants defined earlier, and also for calculating the expansivity gap of these polynomials.

Publications

[1] Uray M.J.: Algebraic number fields and the LLL algorithm. *Under review process*, manuscript: `https://arxiv.org/abs/1810.01634`

[2] Uray M.J.: Characterization of expansive polynomials by special determinants. *Publ. Math. Debrecen* **98/3-4** (2021), pp. 379–399

[3] Uray M.J.: A family of barely expansive polynomials. *Annales Univ. Sci. Budapest., Sect. Math.*, accepted

[4] Bóka D., Burcsi P., Uray M.J.: On the algorithmic complexity of some numeration-related problems. *Publ. Math. Debrecen* **98/1-2** (2021), pp. 65–81

[5] Uray M.J.: On proving inequalities by cylindrical algebraic decomposition. *Annales Univ. Sci. Budapest., Sect. Comp.* **51** (2020), pp. 231–252

References

[6] H. Cohen: A Course in Computational Algebraic Number Theory. *Springer-Verlag Berlin Heidelberg*, 1996

[7] K.O. Geddes, S.R. Czapor, G. Labahn: Algorithms for Computer Algebra. *Kluwer Academic Publishers*, 1992

[8] J.-F. Biasse, C. Fieker, T. Hofmann: On the computation of the HNF of a module over the ring of integers of a number field. *Journal of Symbolic Computation* **80** (2017), pp. 581–615

[9] A.K. Lenstra, H.W. Lenstra, L. Lovász: Factoring Polynomials with Rational Coefficients. *Mathematische Annalen* **261** (1982), pp. 515–534

[10] A. Pethő, M.E. Pohst, Cs. Bertók: On multidimensional Diophantine approximation of algebraic numbers. *Journal of Number Theory* **171** (2017), pp. 422–448

[11] A. Cohn, Über die Anzahl der Wurzeln einer algebraischen Gleichung in einem Kreise. *Mathematische Zeitschrift* **14** (1922), pp. 110–148.