On the complexity of symbolic computation

Joris van der Hoeven
CNRS, École polytechnique, Institut Polytechnique de Paris
Laboratoire d'informatique de l'École polytechnique (LIX, UMR 7161)
Bâtiment Alan Turing, CS35003
1, rue Honoré d'Estienne d'Orves
91120 Palaiseau, France
vdhoeven@lix.polytechnique.fr

ABSTRACT

In this paper, we survey various basic and higher level tasks in computer algebra from the complexity perspective. Particular attention is paid to problems that are fundamental from this point of view and interconnections between other problems.

1 Introduction

Which problems in symbolic computation can be solved reasonably fast? Given a problem, would it be possible to predict how much time is needed to solve it? The area of *computational complexity* was designed to provide answers to such questions. However, symbolic computation has a few particularities that call for extra or separate techniques.

One particularity is that the most prominent algorithms typically get implemented, so we can check theory against practice. A satisfactory complexity theory in our area should at least be able to *explain* or *model* running times that are observed in practice.

However, such a theory should be more than an ennobled tool for benchmarking. We also wish to gain insight into the *intrinsic* complexity of problems, understand how the complexities of different problems are *related*, and identify the most central computational tasks in a given area (these tasks typically require highly optimized implementations).

Traditionally, a distinction is made between the *analysis of algorithms* (how to analyze the complexity of a given algorithm) and *complexity theory* (fitting problems into complexity classes and proving lower bounds). This paper partially falls in the second category, since we will be interested in *classes* of algorithms and problems. Such a broader perspective is particularly useful if you want to implement a new computer algebra system, like MATHEMAGIX in my case [77], or a library for such a system.

However, traditional complexity theory is not very adequate for computer algebra, due to the disproportionate focus on lower bounds and coarse complexity classes. Fortunately, as will be argued in Section 3, we already have many ingredients for developing a more meaningful complexity theory for our area. I found it non-trivial to organize these ingredients into a "grand theory". In this survey, I will therefore focus on concrete examples that reflect the general spirit.

The first series of examples in Section 4 concerns basic operations like multiplication, division, gcds, etc. for various algebras. We will see how various basic complexities are related and how some of them can be derived using general techniques.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
(Please declare conferenceinfo, CopyrightYear, *and* crdata *in your preamble, following ACM guidelines: http://www.acm.org/sigs/publications/proceedings-templates).*
Copyright 20XX ACM XXX-X-XXXX-XXXX-X/XX/XX ...$15.00.

In Section 5, we will turn to the higher level problem of polynomial system solving. My interest in this problem is more recent and driven by complexity considerations. I will survey advantages and limitations of the main existing approaches and highlight a few recent results.

2 Two extremist views on complexity theory

2.1 Traditional complexity theory

Standard theoretical computer science courses on computability and complexity theory [101] promise insight into two questions:

Q1. What can we compute?

Q2. What can we compute efficiently?

Whereas there is broad consensus about the definition of computability (Church' thesis), it is harder to give a precise definition of "efficiency". For this, traditional complexity theory heavily relies on *complexity classes* (such as **P**, **NP**, etc.) in order to describe the difficulty of problems. In principle, this provides some flexibility, although many courses single out **P** as "the" class of problems that can be solved "efficiently".

Now the question **Q2** is actually too hard for virtually all interesting problems, since it involves proving notoriously difficult lower bounds. Our introductory course goes on with the definition of *reductions* to relate the difficulties of different problems: if problem A reduces in polynomial time to an NP-complete problem B, then A is NP-complete as well. This allows us to use NP-completeness as a surrogate for "lower bound".

then it is in NP…

A big part of complexity theory is then devoted to proving such hardness results: the emphasis is on what *cannot* be computed fast; understanding which problems can be solved efficiently (and *why* and *how*) is less of a concern (this is typically delegated to other areas, such as *algorithms*).

There are numerous reasons why this state of the art is unsatisfactory for symbolic computation. At best, negative hardness results may be useful in the sense that they may prevent us from losing our time on searching for efficient solutions. Although:

"You prove that problems are hard and I write computer programs that solve them." (*Richard Jenks [81]*)

For instance, deciding whether a system of polynomial equations over $\mathbb{F}_2$ has *a* solution is a typical NP-complete problem, since it is equivalent to SAT [24]. At the same time, the computer algebra community has a long and successful history of solving polynomial systems [16, 29], at least for *specific instances*. From a theoretical point of view, it was also proved recently that *all* solutions of a generic dense system of polynomial equations over $\mathbb{Z}$ can be computed in expected quasi-optimal time in terms of the expected output size; see [75] and Section 5.5 below.

In the past decade, there has been a lot of work to understand such paradoxes and counter some of the underlying criticism. For instance, the specific nature of certain input instances is better taken into account via, e.g., parametric or instance complexity [108]. Complexity classes with more practical relevance than **P** have also received wider attention [117].

But even such upgraded complexity theories continue to miss important points. First of all, lower bound proofs are rarely good for applications: for the above polynomial time reduction of problem A to the NP-complete problem B, *any* polynomial time reduction will do. If we are lucky to hit an easy instance of A, such a reduction typically produces an intractable instance of B. In other words: negative reductions do not need to be efficient.

Secondly, the desire to capture *practical* complexity via *asymptotic* complexity classes [81, Cook's (amended) thesis] is unrealistic: even a constant time algorithm (like any algorithm on a finite computer which terminates) is not necessarily "practical" if the constant gets large; constants are essential in practice, but hidden through asymptotics.

A more subtle blind spot concerns the framework of complexity classes itself, which has modeled the way how several generations have been thinking about computational complexity. Is it really most appropriate to put problems into pigeonholes **P**, **NP**, **EXP**, **L**, etc.? Sorting and matrix multiplication are both in **P**; does this mean that these problems are related? In computer algebra we have efficient ways to reduce division and gcds to multiplication [35, 15]; the complexities of interpolation, multipoint evaluation, factorization, and root finding are also tightly related (see Section 5); a good complexity theory should reflect such interconnections.

2.2 Take your stopwatch

Implementors tend to have a more pragmatic point of view on complexity:

"Do you want to know whether your algorithm is faster? Take your stopwatch and run it." *(Mark van Hoeij)*

It is true that a complexity-oriented approach to the development of algorithms (as advocated by the author of these lines) may be overkill. For instance, there has been a tendency in computer algebra to redevelop a lot of linear algebra algorithms to obtain complexities that are good in terms of the exponent $\omega < 2.3729$ of matrix multiplication [34]. But ω stays desperately close to 3, in practice.

Yet, there are obvious pitfalls when throwing all complexity considerations overboard: your stopwatch will not predict whether you should take a cup of coffee while your algorithm runs, or whether you should rather go to bed.

What one often needs is a rough educated idea about expected running times. Such a relaxed attitude towards complexity is omnipresent in neighboring areas such as numerical analysis. The popular textbook [104] is full with assertions like "the LU-decomposition [...] requires about $^1/_3 N^3$ executions [...]" (end of Section 2.3.1), "the total operation count for both eigenvalues and eigenvectors is $\sim 25 n^3$" (Section 11.7), etc.

Another obvious problem with benchmarks is that experiments on computers from fifty years ago may be hard to reproduce today; and they probably lost a lot of their significance with time.

But there are also less obvious and more serious problems when your stopwatch is all you got. I hit one such problem during the development of relaxed power series arithmetic [52, 54]. Polya showed [103] that the generating series of the number of stereoisomers with molecular formula $C_nH_{2n+1}OH$ satisfies the equation

$$s(z) = 1 + \frac{1}{3} z (s(z)^3 + s(z^3)).$$

Many terms of this series can be computed using relaxed arithmetic. However, when doing so for rational coefficients, timings were much slower than expected. The reason was that GMP [39] used a naive algorithm for large gcd computations at that time. This came as a big surprise to me: if I had just tried the new algorithms with a stopwatch in my hand, then I probably would have concluded that they were no good!

Study	1997	2004	2012	2017	2021
None	53%	47%	27%	28%	37%
Benchmarks	17%	30%	22%	24%	24%
Complexity analysis	22%	9%	27%	39%	29%
Both	8%	14%	24%	9%	10%

Table 1. Statistics for the number of papers in ISSAC proceedings that contain benchmarks, a complexity analysis, nothing of this kind, or both.

This was really a serious problem. Around the same period, at every computer algebra meeting, there would be an afternoon on fast linear algebra without divisions (variants of Bareiss' algorithm). This research direction became pointless when GMP eventually integrated a better algorithm to compute gcds.

When implementing your own algorithm in a modern computer algebra system, you typically rely on *a lot* of other algorithms, some of which may even be secret in proprietary systems. You also rely on very complex hardware with multi-core SIMD processors and many levels of cache memory. One of the aims of *complexity theory* as opposed to the analysis and benchmarking of a single *algorithm* should be to get a grip on the broad picture.

A better global understanding might guide future developments. For instance, working in algebraic extensions has bad press in computer algebra for being slow; "rational" algorithms have been developed to counter this problem. But is this really warranted from a complexity point of view? Could it be that algebraic extensions simply are not implemented that well?

In symbolic computation, it is also natural to privilege symbolic and algebraic methods over numeric or analytic ones, both because many of us simply like them better *and* because we have more efficient support for them in existing systems. But maybe some algebraic problems could be solved more efficiently using numerical methods. What about relying more heavily on homotopy methods for polynomial system solving? What about numerical differential Galois theory [23]?

3 Symbolic computation

3.1 Complexity at ISSAC

How does the ISSAC community deal with complexity issues? In Table 1, I compiled some statistics for actual papers in the proceedings, while limiting myself to the years 1997, 2004, 2012, 2017, and 2021. I checked whether the papers contained benchmarks, a complexity analysis, both, or nothing at all.

It is clear from Table 1 that the ISSAC community cares about complexity issues. Many papers contain a detailed complexity analysis. For high-level algorithms with irregular complexities, authors often provide benchmarks instead. Besides providing interesting insights, one may also note that "objective" benchmarks and complexity results are frequently used to promote a new algorithm, especially when there are several existing methods to solve the same problem.

Only a few papers mention traditional complexity classes and lower bounds are rarely examined through the lens of computational hardness. Instead, it is typically more relevant to determine the cost and the size of the output for generic instances. For instance, instead of proving that "polynomial system solving is hard", one would investigate the complexity of solving a generic system in terms of the Bézout bound (which is reached for such a system).

Most theoretical complexity results are stated in the form of upper bounds. Both algebraic and bit-complexity models are very popular; occasionally, bounds are stated differently, e.g. in terms of heights, condition numbers, probability of success, etc.

One interesting characteristic of algorithms in symbolic computation is that their complexities can often be expressed in terms of the basic complexities of multiplying n-bit integers, degree d polynomials, and $r \times r$ matrices. Several papers even use special notations $\mathsf{I}(n)$, $\mathsf{M}(d)$, and r^{ω} for these complexities. In a similar manner, number theorists introduced

$$L_n[\alpha, c] := \mathrm{e}^{(c+o(1))(\log n)^{\alpha}(\log\log n)^{1-\alpha}}$$

in order to express the complexities of fast algorithms for integer factorization and discrete logarithms. We will return to these observations in Section 3.2 below.

In the past decade, there have been many developments "beyond the worst-case" analysis of algorithms [108]. To which extent are these developments relevant to the ISSAC community? For instance, a hard problem may become easier for particular input *distributions* that are *meaningful* in practice. It is important to describe and understand such distributions more precisely. Surprisingly, very few papers in our area pursue this point of view. On the other hand, there are many lists of "interesting" problems for benchmarking. Conversely, only the last page 660 of [108] contains a table with experimental timings.

In my personal research, I also encountered various interesting algorithms that scream for yet other non-traditional analyses: amortized complexity (fast algorithms modulo precomputations [73]), conjectured complexity (dependence on plausible but unproven conjectures [49]), heuristic complexity [69], irregular complexity (various operations over finite fields $\mathbb{F}_{p^{\kappa}}$ can be done faster if κ is composite or smooth [67, 76]).

3.2 Fundamental complexities

Hardy already noted that many natural orders of growth can be expressed using exp-log functions [43]; this in particular holds for the asymptotic complexities of many algorithms. Instead of searching for "absolute" complexity bounds, it has become common to express complexities in terms of other, more "fundamental" complexities such as $\mathsf{I}(n)$, $\mathsf{M}(d)$, r^{ω}, and $L_n[\alpha, c]$ that were introduced above.

In a similar way as traditional complexity theory is organized through complexity classes **P**, **NP**, **EXP**, **L**, etc., I think that fundamental complexities can play a similar role for symbolic computation and beyond.

Whereas complexity classes give a rough idea about how fast certain problems can be solved, fundamental complexities provide more detailed information such as "the problem essentially reduces to linear algebra".

In fact, the language of complexity bounds (especially when enriched with notations such as $\mathsf{I}(n)$, $\mathsf{M}(d)$, and r^{ω}) is more expressive and precise than the language of complexity classes: whereas

$$\mathsf{PRIMES} \in \mathbf{P}$$

vaguely states that primality can be checked "fast",

$$\mathsf{GCD}(d) = O(\mathsf{M}(d) \log d) \qquad (1)$$

states that computing gcds essentially reduces to multiplication, plausibly using a dichotomic algorithm.

Another advantage of (re-)organizing complexity theory in this direction is that fundamental complexities are *hardware-oblivious*: the bound (1) holds on Turing machines, for algebraic complexity models, on quantum computers, as well as on many parallel machines. Deciding whether integer factorization can be done in polynomial time may lead to different outcomes on Turing machines and quantum computers.

With the help of fundamental complexities, even constant factors can be made more precise. For instance, [35, Exercise 11.6] indicates how to prove that

$$\mathsf{GCD}(d) \leqslant \left(\frac{10}{\log 2} + o(1)\right) \mathsf{M}(d) \log d.$$

This often makes it possible to predict running times very accurately, provided we know the performance of a few basic algorithms. Implementors can then optimize these basic algorithms.

Fundamental algorithms may also be implemented in hardware. For instance, Apple's new M1 processor contains a matrix coprocessor. With suitable hardware, it might very well be possible to "achieve" $\omega = 2$ for many real world applications.

The question remains which complexities are worthy to be dubbed "fundamental". This question is actually a productive common thread whenever you want to understand complexity issues in some area. It asks you to identify the most fundamental algorithmic building blocks. Sometimes, this requires the introduction of a new fundamental complexity (such as modular composition or multi-point evaluation for polynomial system solving; see Section 5 below). Sometimes, you may find out that your area reduces to another one from an algorithmic point of view. For instance, computations with the most common Ore operators essentially reduce to polynomial linear algebra.

3.3 Back to our stopwatch

Asymptotic complexity bounds can be treacherous from a practical point of view. The most notorious example is matrix multiplication: it has been proved that $\omega < 2.3729$ [34], but I am not aware of any implementation that performs better than Strassen's algorithm [115] for which $\omega = \log_2 7 \approx 2.807$. A more interesting example is Kedlaya–Umans' recent algorithm for modular composition [84]. We were unable to make it work fast in practice [70, Conclusion], but it is unclear whether variants of the algorithm might allow for efficient implementations.

Whereas hardness results dissuade implementors from searching overly efficient solutions for certain problems, the main practical interest of the above complexity bounds is to dissuade complexity theorists from trying to prove overly pessimistic lower bounds.

One useful indicator for practical differences in performance is the *crossover point*: if A is asymptotically faster than B, for which size $n_{A,B}$ of the input does A actually become faster? Curiously, crossover points are often mentioned *en passant* when discussing benchmarks, but they have not really been the subject of systematic studies: does anyone know what is the crossover point between naive matrix multiplication and any of the asymptotically fast algorithms with $\omega \leqslant 2.5$?

It is illuminating to study the complexities of A and B *near* the crossover point. For instance, for Strassen's algorithm versus naive matrix multiplication, assume that the crossover point is $n_{A,B} \approx 1000$. Then this means that we need a matrix of size 32000×32000 in order to gain a factor two with Strassen's method: for $\omega = \log_7 2$, we have $(2^5)^3 / (2^5)^{\omega} = 2^{5(3-\omega)} \approx 1.95$.

Now there are many recent bounds in the recent computer algebra literature of the form $T(n) = \tilde{O}(n^{a\omega + b})$. Here the soft-O notation $\tilde{O}(f)$ stands for $O(f (\log f)^{O(1)})$ and serves to "discard" all logarithmic factors. Although I think that it is a good practice to state complexity bounds in terms of ω (for instance, ω might be lower for sparse or structured matrices), I suspect that the discarded logarithmic factors are often far more significant.

Algebra	Complexity	Reference, validity
$\mathbb{Z}$	$O(n \log n)$	[48], bit complexity
$\mathbb{K}[x]$	$O(d \log d \log \log d)$	[113, 111, 19], algebraic
	$O\left(d \log d\, 4^{\log^* d}\right)$	[47], char $\mathbb{K} > 0$
	$O(d \log d)$	[49], char $\mathbb{K} > 0$, conjectured
$\mathbb{K}^{r \times r}$	$O(r^{\omega})$, $\omega < 2.3729$	[34], algebraic

Table 2. Best known complexity bounds for multiplying integers, polynomials, and matrices.

In this paper, I focus on sequential bit-complexity [101] and algebraic complexity [17]. Of course, modern computers come with complex cache hierarchies and they are highly parallel (distributed over networks, multi-core, and with wide SIMD instructions). A model for measuring the cost of cache misses was proposed in [33]. For a past ISSAC tutorial on parallel programming techniques in computer algebra, I refer to [94]. The Spiral project [106] is also noteworthy for using computer algebra to generate parallel code for low level transforms such as FFTs.

4 Basic arithmetic

Computer algebra systems are usually developed in layers, as are mathematical theories. The bottom layer concerns numbers. The next layers contain basic data structures such as univariate polynomials, dense matrices, symbolic expressions, and so on. Gröbner bases, symbolic integration, etc. can be found in the top layers.

My mental picture of a complexity theory for computer algebra roughly follows the same structure. The aim is to understand the complexity issues layer by layer, starting with the lowest ones.

4.1 Multiplication

In order to figure out the cost of basic arithmetic operations, multiplication is the central operation to analyze first. We have already encountered the fundamental complexities $\mathsf{I}(n)$, $\mathsf{M}(d)$, and r^{ω}, for which the best known bounds are summarized in Table 2.

Multiplication is fundamental in two ways. On the one hand, many other operations can efficiently be reduced to multiplication. On the other hand, the techniques that allow for fast multiplication are typically important for other operations as well. In Table 3, I summarized the complexity of multiplication for a few other important algebras.

Other noteworthy bounds in the same spirit concern truncated multiplication [96, 42, 46], middle products [41, 15], multivariate power series [88], symmetric polynomials [66], rectangular matrix multiplication [79, 86], and sparse differential operators [36].

Technically speaking, many algorithms for fast multiplication rely on *evaluation-interpolation* (or, more generally, they rewrite the problem into a simpler multiplication problem for another algebra). The complexity of such algorithms can be expressed in terms of the number of evaluation points and the cost of conversions.

Algebra	Complexity	Notes
$\mathbb{Z}_p$	$\mathsf{I}(n)\, \mathrm{e}^{O(\sqrt{\log \log n})}$	[59], relaxed
$\mathbb{K}[[z]]$	$\mathsf{M}(d)\, \mathrm{e}^{O(\sqrt{\log \log d})}$	[59], relaxed
$\mathbb{K}^{r \times r}[x]$	$O(r^2\, \mathsf{M}(d) + r^{\omega} d)$	[12]
$\mathbb{K}[x, \partial_x]$	$O(r\, \mathsf{M}(d) \log d + r^{\omega-1} d)$	[6], $d \geqslant r = \deg_{\partial}$
	$O(d\, \mathsf{M}(r) \log r + d^{\omega-1} r)$	[6], $r \geqslant d$
$\mathbb{L} \mid \mathbb{K}$ tower	$\mathsf{M}(d)\, \mathrm{e}^{O(\sqrt{\log d})}$	[68], $d = [\mathbb{L} : \mathbb{K}]$
$\mathbb{K}[\boldsymbol{x}]$	$O(\mathsf{M}(s))$	[69, 64], sparse, heuristic

Table 3. Best known complexity bounds for multiplication in various algebras. In the last bound, s stands for the total size of the supports of the input and output.

Operation	Complexity	References
Quotient+remainder	$\sim 2\, \mathsf{I}(n)$	[57]
Reciprocal	$\sim {}^{13}/_{9}\, \mathsf{I}(n)$	[45]
Square root	$\sim {}^{4}/_{3}\, \mathsf{I}(n)$	[45]
Base conversion Amortized CRT	$O\left(\mathsf{I}(n) \frac{\log n}{\log \log n}\right)$	[61]
Gcd, lcm	$O(\mathsf{I}(n) \log n)$	[110, 93]
CRT	$O(\mathsf{I}(n) \log n)$	[32, 92]
Exponential Logarithm	$O(\mathsf{I}(n) \log n)$	[13]
Evaluation of holonomic functions	$O(\mathsf{I}(n) \log^2 n)$	[22, 53, 91]

Table 4. Complexity bounds for various basic operations on n-bit integers and/or floating point numbers. The first four bounds assume the FFT-model.

For instance, given an *evaluation-interpolation scheme* for degree d polynomials, let $\mathsf{N}(d)$ and $\mathsf{E}(d)$ denote the number of evaluation points and the cost of evaluation/interpolation. Then the complexity of multiplying two matrices of degree d in $\mathbb{K}[x]^{r \times r}$ becomes

$$\mathsf{N}(d)\, r^{\omega} + 3\, \mathsf{E}(d)\, r^2.$$

The evaluation-interpolation scheme should carefully be selected as a function of d and r, with a trade-off between the efficiency of the conversions and the compactness of the evaluated polynomials.

There exist numerous important schemes: fast Fourier transforms [25], Karatsuba and Toom-Cook transforms [83, 116], Chinese remaindering [26, 61], Nussbaumer (or Schönhage–Strassen) polynomial transforms [113, 99], truncated Fourier transforms [55], Chudnovsky^2 evaluations and interpolations [21], and so on.

4.2 Basic operations

For other basic arithmetic operations like division, square roots, gcds, matrix decompositions, exp, log, etc., one may proceed in two stages: we first seek an efficient reduction to multiplication and then further optimize the method for particular multiplication schemes.

For instance, operations such as division and square root, but also exponentials of formal power series, can be reduced efficiently to multiplication using Newton's method [14]. When using such an algorithm in combination with FFT-multiplication, some of the FFTs are typically redundant. Constant factors can then be reduced by removing such redundancies [7, 8]. Note that Shoup's NTL library is an early example of a library that makes systematic use of specific optimizations for the FFT-model [114].

One may go one step further and modify the algorithms such that even more FFTs are saved, at the cost of making the computations in the FFT-model a bit more expensive. This technique of *FFT trading* leads to even better asymptotic constant factors or complexities [57, 45, 44, 61].

Table 4 contains a summary of the best known asymptotic complexity bounds for various basic operations on integers and/or floating point numbers. Similar bounds hold for polynomials and truncated power series. The first three bounds were actually proved for polynomials and/or series, while assuming the FFT-model: one degree d multiplication is equivalent to six DFTs of length d.

CRT stands for the Chinese remainder theorem; conversions in either direction have the same softly linear complexity. This was originally proved for polynomials, but also holds for integer moduli [15, Section 2.7]. For *amortized* CRTs, the moduli are fixed, so quantities that only depend on the moduli can be precomputed. A *holonomic function* is a function that satisfies a linear differential equation over $\bar{\mathbb{Q}}[x]$, where $\bar{\mathbb{Q}}$ is the field of algebraic numbers. Many special functions are of this type.

Some bounds for transcendental operations are better for power series than for numbers. For instance, exponentials can be computed in time $\sim {}^{13}\!/_{6}\,\mathsf{M}(d)$ modulo $O(z^d)$. Using *relaxed* or *online* arithmetic, when coefficients are fed one by one, solving power series equations is as efficient as evaluating them [54, 63]. For instance, in the Karatsuba model $g = \exp f$ can be computed in time $\sim\mathsf{M}(d)$ using $g = \int f' g$. In the FFT-model, one has to pay a small, non-constant price for relaxed multiplication (see Table 3).

The numerical evaluation of analytic functions reduces into power series computations and the computation of effective error bounds. This can be used to solve many problems from numerical analysis with high precision. For instance, the reliable integration of an algebraic dynamical system between two fixed times and with a variable precision of n bits takes $O(\mathsf{I}(n^2))$ bit operations [58]. See [80, 90] for other software packages for reliable numerical computations with high precision.

Similar tables as Table 4 can be compiled for other algebras. For instance, the inverse of an $r \times r$ matrix with entries in a field $\mathbb{K}$ can be computed in time $O(r^{\omega})$ [115]. If $\mathsf{SM}(d, r)$ stands for the complexity of multiplying two operators in $\mathbb{K}[x, \partial]$ of degree $\leqslant d$ in x and order $\leqslant r$ in ∂, then exact right division of two such operators can be done in time $O(\mathsf{SM}(d, r) \log d)$ [62].

4.3 Sparse interpolation

One big enemy of symbolic computation is *intermediate expression swell*. This phenomenon ruins the complexity of many algorithms. For instance, let M be an $r \times r$ matrix with formal entries $a_{i,j}$ and consider the computation of $((M^{-1})^{\top})^{-1}$. For $n \geqslant 20$, the intermediate expression for M^{-1} is prohibitively large, when fully expanded as a rational function in the parameters $a_{i,j}$. But the end-result $M^{\top}$ is fairly small. The technique of *sparse interpolation* proposes a remedy to this problem.

In general, we start with a blackbox function f, such as the above function that associates $((M^{-1})^{\top})^{-1}$ to $(a_{i,j})_{1 \leqslant i,j \leqslant r}$. The function f is usually given by a DAG that computes one or more polynomials or rational functions that involve many parameters. The DAG should be of reasonable size, so that it can be evaluated fast. In our example, the DAG might use Gaussian elimination to compute the inverses. Then sparse interpolation allows for the efficient symbolic reconstruction of the actual polynomials or rational functions computed by f, whenever these are reasonably small. This is an old topic [105, 5, 18] with a lot of recent progress; see [107] for a nice survey.

Let us focus on the case when f computes a polynomial with rational coefficients. From a complexity perspective, the most important parameters are the size L of f as a DAG, the size s of its support as a polynomial, and the bit size b of the coefficients (the degree and dimension are typically small with respect to $s + b$, so we will ignore them for simplicity). Sparse interpolation goes in two stages: we first determine the support of f, typically using a probabilistic algorithm of Monte Carlo type. We next determine the coefficients.

Now the first stage can be done modulo a sufficiently large prime $p = \Omega(\log s)$, so the complexity hardly depends on b. Using Prony's algorithm [105], this stage first involves $2s$ evaluations of f of cost $O(L\, s\, \mathsf{I}(\log p))$; using Cantor–Zassenhaus for polynomial root finding [20], we next need $O\big(\mathsf{M}_{\mathbb{F}_p}(s) \log^2 s\big) = O(s \log^3 s\, \mathsf{I}(\log p))$ operations to recover the support. Finally, the coefficients can be computed using transposed polynomial interpolation [18, 11], in time $O(\mathsf{I}(s\, b) \log s)$.

The dominant term in the resulting complexity $O(s\,(L + \log^3 s)\,\mathsf{I}(\log p) + \mathsf{I}(s\, b) \log s)$ depends on the asymptotic region we are in. The logarithmic terms are significant. For instance, if $f = \det M$ for the above matrix M, then $s = r!$, $\log s = \Theta(r \log r)$, $L = r^{\omega}$, and $b = O(1)$, whence the Cantor–Zassenhaus step dominates the cost. In [40, 78], it was shown how to save a factor $\Omega(\log s)$ by using the tangent Graeffe method; after that, the evaluation step dominates the cost.

For basic arithmetic operations on sparse polynomials or low dimensional linear algebra with sparse polynomial coefficients, the evaluation cost L can be amortized using multi-point evaluation; this essentially achieves $L = \Theta(\log s)$. In such situations, both Prony's algorithm and the tangent Graeffe algorithm become suboptimal. It then becomes better to evaluate directly at suitable roots of unity using FFTs, which allows for another $\Theta(\log s)$ speed-up, also for the computation of the coefficients. In particular, sparse polynomial multiplication becomes almost as fast as dense multiplication. However, the technical details are subtle and can be found in [69, 64].

5 Polynomial system solving

One central problem in symbolic computation is polynomial system solving. Now the mere decision problem whether there exists *one* solution is a notorious NP-hard problem. However, most algorithms in computer algebra concern the computation of *all* solutions, for various representations of solutions. The number of solutions may be huge, but the cost of finding them is often reasonable with respect to the size of the output.

In this section, we focus on the complete resolution of a generic affine system of n polynomial equations of degrees $d_1, \ldots, d_n$. For definiteness, we assume that the coefficients are in $\mathbb{Z}$, $\mathbb{C}$, or $\mathbb{F}_p$. For simplicity, we also assume that $d_1 = \cdots = d_n = d$. For such a generic system, all solutions are isolated and simple and the number of solutions reaches the Bézout bound $D = d_1 \cdots d_n = d^n$.

There exist several approaches for polynomial system solving:

- Gröbner bases.
- Numeric homotopies.
- Geometric resolution.
- Triangular systems.

We will restrict our exposition to the first three approaches. But we start with a survey of the related problem of multi-point evaluation, followed by an investigation of the univariate case, which calls for separate techniques.

5.1 Multi-point evaluation

Let $\mathbb{K}$ be a field and $\mathbb{A}$ an algebra over $\mathbb{K}$. The problem of multi-point evaluation takes $P \in \mathbb{K}[\boldsymbol{x}] = \mathbb{K}[x_1, \ldots, x_n]$ and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in (\mathbb{A}^n)^N$ as input and should produce $P(\boldsymbol{x}_1), \ldots, P(\boldsymbol{x}_N)$ as output. This problem is closely related to polynomial system solving: given a tentative solution of such a system, we may check whether it is an actual solution using multi-point evaluation. For complex or p-adic solutions, we may also double the precision of an approximate solution through a combination of Newton's method and multi-point evaluation.

Using the technique of remainder trees [35, Chapter 10], it is classical that the univariate case with $\mathbb{A} = \mathbb{K}$ can be solved in softly optimal time $O(\mathsf{M}(d)\, N / d)$ if $N \geqslant d$ and $O(\mathsf{M}(N)\, d / N)$ if $d \geqslant N$.

The special case when $n = N = 1$ and $\mathbb{A}$ is an algebraic extension of $\mathbb{K}$ (typically of degree d) is called the problem of *modular composition*. This problem can be solved in time $O(n^{\varpi} + \sqrt{n}\, \mathsf{M}_{\mathbb{K}}(n))$ using Stockmeyer and Paterson's baby-step-giant-step technique; see [102] and [82, p. 185]. The constant $\varpi > 1.5$ is such that a $\sqrt{n} \times \sqrt{n}$ matrix over $\mathbb{K}$ may be multiplied with another $\sqrt{n} \times n$ rectangular matrix in time $O(n^{\varpi})$. One may take $\varpi < 1.629$ [86, 73]. The same technique can be applied for other multi-point/modular evaluation problems [68, Section 3].

A breakthrough was made in 2008 by Kedlaya and Umans [84], who showed that modular composition and multi-point evaluation with $\mathbb{A} = \mathbb{K}$ are related and that they can both be performed with quasi-optimal complexity. For instance, the complexity of modular composition is $d\,\mathrm{e}^{O(\sqrt{\log d \log \log d})}$. Unfortunately, their algorithms do not seem to be efficient in practice [70, Conclusion]; being non-algebraic, their approach is also not generally applicable in characteristic zero.

In the past decade, Grégoire Lecerf and I have investigated several more practical alternatives for Kedlaya-Umans' algorithms. Over the integers and rationals, modular composition can be done in softly optimal time for large precisions [71]. Over finite fields $\mathbb{F}_q$ with $q = p^\lambda$ and p prime, it helps when λ is composite [67]; if λ is smooth, then composition modulo a fixed irreducible polynomial of degree λ can even be done in softly optimal time.

There has also been progress on the problem of amortized multi-point evaluation, when the evaluation points are fixed [74, 97, 72]. This has culminated in a softly optimal algorithm for any fixed dimension [73]. It would be interesting to know whether general multi-point evaluation can be reduced to the amortized case.

5.2 Univariate polynomials

Consider the problem of computing the complex roots $z_1, \ldots, z_d$ of a square-free polynomial $P \in \mathbb{C}[z]$. Obviously, this problem is equivalent to the problem to factor P over $\mathbb{C}$.

If we have good approximations of the roots, then we may use Newton's method to double the precision of these approximations. Provided that our bit-precision b is sufficiently large, this can be done for all roots together using multi-point evaluation, in softly optimal time $O(\mathsf{I}(b\,d) \log d)$. Our problem thus contains two main difficulties: how to find good initial approximations and what to do for small bit-precisions?

Based on earlier work by Schönhage [112], the root finding problem for large precisions was solved in a softly optimal way by Pan [100]. Assuming that $|z_i| \leqslant 1$ for all i, he essentially showed that P can be factored with b bits of precision in time $O(d \log^2 d\,(\log^2 d + \log b)\,\mathsf{I}(b)) = \tilde{O}(d\,(d+b))$. In particular, the soft optimality occurs as soon as $b = \Theta(d)$.

The remaining case when $b = o(d)$ was solved recently by Moroz [95]. He shows how to isolate the roots of P in time $\tilde{O}(d\,(b + \log \kappa))$, where κ is a suitable condition number. A crucial ingredient of his algorithm is fast multi-point evaluation for small b. Moroz shows that this can be done in time $\tilde{O}(d\,b)$, by improving drastically on ideas from [56, Section 3.2]. A refined analysis shows that his algorithm can be made to run in time $O(\mathsf{I}(d\,b) \log d)$ when $b > \log d$.

For polynomials $P \in \mathbb{Q}[x]$, one may also ask for an algorithm to factor P over $\mathbb{Q}$ instead of approximating its roots in $\mathbb{C}$. The famous LLL-algorithm provided the first polynomial-time algorithm for this task [89]. A practically more efficient method was proposed in [50]. For subsequent improvements on the complexity of lattice reduction and its application to factoring, see [51, 98].

When P has coefficients in the finite field $\mathbb{F}_q$ with $q = p^\lambda$ and p prime, the best current general purpose factorization algorithm is (Las-Vegas) probabilistic and due to Cantor–Zassenhaus [20]; it has (expected) complexity $O(\mathsf{M}_q(d) \log q \log d)$, where $\mathsf{M}_q(d)$ stands for the cost of multiplying two polynomials of degree d over $\mathbb{F}_q$ (assuming a plausible number-theoretic conjecture, we have $\mathsf{M}_q(d) = O(d \log q \log(d \log q))$). Note that the bit-complexity of the Cantor–Zassenhaus algorithm is quadratic in $\log q$. If P splits over $\mathbb{F}_q$, then the Tangent-Graeffe algorithm is often more efficient [40, 78]. In particular, if $q-1$ is smooth, then the complexity drops to $O(\mathsf{M}_q(d) \log d)$.

If λ is large, then factorization over $\mathbb{F}_q$ can be reduced efficiently to modular composition [82]. This yields the bound $d^{1.5+o(1)} \log^{1+o(1)} q + \tilde{O}(d \log^2 q)$ when using Kedlaya–Umans' algorithm for modular composition. If λ is composite, then [67] describes faster algorithms for modular composition. If λ is smooth and d is bounded, then factorization can even be done in softly optimal time $\tilde{O}(d \log q)$ [76].

5.3 Gröbner bases

Macaulay matrices are a classical tool to reduce the resolution of generic polynomial systems to linear algebra. For n affine equations of degree d, the complexity of this method is $O\!\left(n\,\delta \binom{n+\delta}{\delta}^{\omega}\right)$, where $\delta = n\,d + 1 - n$ [10, Théorème 26.15]. For a fixed dimension n, this bound becomes $O((d^n)^{\omega + O(1/n)})$. Many modern Gröbner basis implementations essentially use this method, while avoiding the computation of rows (or columns) that can be predicted to vanish. In particular, Faugère's F5 algorithm [30] can be formulated in this way [2] and runs in time $(d^n)^{3+o(1)}$, uniformly in d and n. For a generic system of n equations of degree d, it follows that we can compute a Gröbner basis with quasi-cubic complexity in terms of the number d^n of solutions.

Practical Gröbner basis computations are most efficient with respect to graded monomial orderings. But solutions of the system are more convenient to read from Gröbner bases with respect to lexicographical orderings. Changes of orderings for zero-dimensional systems can be performed efficiently using the FGLM algorithm [28]. For a generic system, the complexity of a recent optimization of this method is $\tilde{O}(D^\omega)$, where D is the number of solutions [31].

But is the computation of a Gröbner basis intrinsically a linear algebra problem? After all, gcds of univariate polynomials can be computed in softly optimal time. It turns out that better algorithms also exist in the bivariate case. One subtlety is that the standard representation of a Gröbner basis for a generic system with $d_1 = d_2 = d$ already requires $O(d^3)$ space with respect to a graded monomial ordering. In [65], it was shown that generic bivariate Gröbner bases can be computed in softly optimal time $\tilde{O}(d^2)$ when using a suitable, more compact representation.

Some softly optimal techniques also exist also for higher dimensions, such as fast relaxed reduction [60] and heterogeneous Gröbner bases [73]. It is unclear whether these techniques can be used to break the linear algebra barrier.

Remark. Let $I = (f_1, f_2)$ be the ideal generated by two generic bivariate polynomials of degree d. The fast algorithm from [65] for computing a Gröbner basis of I also allows the resultant of f_1 and f_2 to be computed faster. Until recently, complexity bounds for this task were softly cubic $\tilde{O}(d^3)$. Over fields with finite characteristic, the complexity drops to $d^{1+o(1)}$, using fast bivariate multi-point evaluation [84]. For general coefficient fields, Villard [118] recently gave an algebraic algorithm for a slightly different problem that runs in time $d^{3-1/\omega+o(1)}$.

5.4 Numeric homotopies

When $\mathbb{K} \subseteq \mathbb{C}$, another popular method to solve polynomial systems is to use numeric homotopies [3]. Given n generic equations $f_1, \ldots, f_n$ of degree d, the idea is to find equations $g_1, \ldots, g_n$ of the same degrees that can be solved easily and then to continuously deform the easy equations in the target equations while following the solutions. For instance, one may take $g_k = z_k^d - c_k$ and $h_{k,t} = (1-t)\,f_k + g_k$ for random constants $c_1, \ldots, c_n$ and follow the solutions of $h_{1,t}(z) = \cdots = h_{n,t}(z)$ from $t = 1$ until $t = 0$.

The theoretical complexity of such methods has extensively been studied in the BSS model [9], in which arithmetic operations on real numbers with arbitrary precision can be done with unit cost. This model is suitable for well conditioned problems from classical numerical analysis, when all computations can be done using machine precision. This holds in particular for random polynomial systems. It has been shown in [4] that one solution path can then be followed in expected average time $O(d^{3/2} n N)$, where $N \leqslant n\binom{n+d}{d}$ is the number of coefficients of $f_1, \ldots, f_n$ for the dense encoding. This bound has been lowered to $O(d^{3/2} n N^{1/2})$ in [1]. A theoretical deterministic algorithm that runs in average polynomial time was given in [85].

In practice, the observed number of homotopy steps seems to grow fairly slowly with n and d for random systems (a few hundred steps typically suffice), so the above bounds are pessimistic. However, we are interested in the computation of all D solutions. When assuming that the average number of steps remains bounded, this gives rise to a complexity $O\!\left(n\binom{n+d}{d} d^n\right)$. If we also take into account the required bit precision b, then the bit complexity becomes $O\!\left(n\binom{n+d}{d} d^n \mathsf{I}(b)\right)$. Fortunately, the bulk of the computation can usually be done with fixed precision and we may directly use Newton's method to repeatedly double the precision at the end $t=0$. If we were able to develop a softly optimal algorithm for numeric multivariate multi-point evaluation, then the complexity would be $\tilde{O}\!\left(\left(\binom{n+d}{d} + \mathsf{I}(b)\right) d^n\right)$.

Numeric homotopies perform particularly well for random systems, which behave essentially like generic ones. However, the existing software is less robust than algebraic solvers for degenerate systems, especially in presence of solutions with high multiplicities. It remains a practical challenge to develop more robust homotopy solvers for general systems and a theoretical challenge to understand the complexity of this problem.

5.5 Geometric resolution

The technique of *geometric resolution* was developed in [38, 37] and further perfected in [87, 27]. It works over arbitrary fields $\mathbb{K}$ and polynomials that are given by a DAG of size L. For simplicity, we assume that our system is generic. After a random linear change of variables, the idea is to successively solve the systems

$$\begin{aligned} &f_1(z_1, 0, \ldots, 0) = 0 \\ &f_1(z_1, z_2, 0, \ldots, 0) = f_2(z_1, z_2, 0, \ldots, 0) = 0 \\ &\vdots \end{aligned}$$

in the form $z_k = a_k(u) / q'(u)$, where u is a formal parameter that satisfies $q(u)=0$ for $q \in \mathbb{K}[u]$. This representation of solutions is called the *Kronecker representation*. It is also a *rational univariate representation* [109] with a special type of denominator.

Let us focus on the last step which dominates the cost. The solutions $z_k = a_k(u) / q'(u)$ of the system

$$f_1(z_1, \ldots, z_{n-1}, 0) = \cdots = f_{n-1}(z_1, \ldots, z_{n-1}, 0) = 0$$

are first lifted into solutions $z_k = a_{k,t}(u) / q_t'(u)$ of the system

$$f_1(z_1, \ldots, z_{n-1}, t) = \cdots = f_{n-1}(z_1, \ldots, z_{n-1}, t) = 0.$$

We next intersect with the hypersurface $f_n(z_1, \ldots, z_{n-1}, t) = 0$. It turns out that it suffices to work with power series in t modulo $O(t^{D+1})$. Then the intersection step gives rise to the computation of a large resultant, which can be done in time $\tilde{O}(D^2/d)$. Altogether it is shown in [87] that the algorithm requires $L\,\tilde{O}(D^2)$ expected operations in $\mathbb{K}$.

Now $L = n\binom{n+d}{n}$ for a dense system of n equations of degree d. For large n and fixed d, we observe that $L\,\tilde{O}(D^2) = \tilde{O}(D^2)$. However, for fixed n and large d, the bound $L\,\tilde{O}(D^2) = \tilde{O}(D^3)$ is only softly cubic. Using variants of Kedlaya–Umans' algorithms for fast multi-point evaluation, the cost of the polynomial evaluations can be amortized, which leads to a quasi-quadratic bound $D^{2+o(1)}$ over finite fields [75]. When working over rational numbers, the bit precision generically grows with D as well and the output is of size $O(D^2)$; in this case, the complexity bound actually becomes quasi-optimal [75, Theorem 6.11].

5.6 Conclusion

Can a complexity-driven approach help us to solve polynomial systems faster? In the above sections, we have seen that such an approach naturally leads to other questions:

- What is the complexity of multivariate multi-point evaluation?
- How can we take advantage of fast polynomial arithmetic?
- How does bit complexity rhyme with numerical conditioning and clusters of solutions?

These questions are interesting for their own sake and they have indeed triggered at least some of the recent progress.

Acknowledgments. I wish to thank Amir Hashemi and Fatima Abu Salem for their careful reading and suggestions.

BIBLIOGRAPHY

[1] D. Armentano, C. Beltrán, P. Bürgisser, F. Cucker, and M. Shub. Condition length and complexity for the solution of polynomial systems. *J. FOCM*, 16:1401–1422, 2016.

[2] M. Bardet, J.-C. Faugère, and B. Salvy. On the complexity of the F5 Gröbner basis algorithm. *JSC*, 70:49–70, 2015.

[3] D. J. Bates, J. D. Hauenstein, A. J. Sommese, and C. W. Wampler. *Numerically Solving Polynomial Systems with Bertini*. SIAM, 2013.

[4] C. Beltrán and L. M. Pardo. Fast linear homotopy to find approximate zeros of polynomial systems. *Found. Comput. Math.*, 11:95–129, 2011.

[5] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proc. ACM STOC '88*, pages 301–309. New York, NY, USA, 1988.

[6] A. Benoit, A. Bostan, and J. van der Hoeven. Quasi-optimal multiplication of linear differential operators. In *Proc. FOCS '12*, pages 524–530. New Brunswick, October 2012. IEEE.

[7] D. J. Bernstein. Removing redundancy in high precision Newton iteration. Available from `http://cr.yp.to/fastnewton.html`, 2000.

[8] D. J. Bernstein. *Fast multiplication and its applications*, pages 325–384. Mathematical Sciences Research Institute Publications. Cambridge University Press, United Kingdom, 2008.

[9] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation*. Springer-Verlag, 1998.

[10] A. Bostan, F. Chyzak, M. Giusti, G. Lecerf, B. Salvy, and É. Schost. *Algorithmes efficaces en calcul formel*. Auto-édition, 2017.

[11] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *Proc. ISSAC '03*, pages 37–44. Philadelphia, USA, August 2003.

[12] A. Bostan and É. Schost. Polynomial evaluation and interpolation on special sets of points. *J. of Complexity*, 21(4):420–446, August 2005. Festschrift for the 70th Birthday of Arnold Schönhage.

[13] R. P. Brent. Fast multiple-precision evaluation of elementary functions. *Journal of the ACM*, 23(2):242–251, 1976.

[14] R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *Journal of the ACM*, 25:581–595, 1978.

[15] R. P. Brent and P. Zimmermann. *Modern Computer Arithmetic*. Cambridge University Press, 2010.

[16] B. Buchberger. *Ein Algorithmus zum auffinden der Basiselemente des Restklassenringes nach einem null-dimensionalen Polynomideal*. PhD thesis, University of Innsbruck, 1965.

[17] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*. Springer-Verlag, Berlin, 1997.

[18] J. Canny, E. Kaltofen, and Y. Lakshman. Solving systems of nonlinear polynomial equations faster. In *Proc. ISSAC '89*, pages 121–128. Portland, Oregon, July 1989.

[19] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28:693–701, 1991.

[20] D. G. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Math. Comp.*, 36(154):587–592, 1981.

[21] D. V. Chudnovsky and G. V. Chudnovsky. Algebraic complexities and algebraic curves over finite fields. *J. of Complexity*, 4:285–316, 1988.

[22] D. V. Chudnovsky and G. V. Chudnovsky. Computer algebra in the service of mathematical physics and number theory (Computers in mathematics, Stanford, CA, 1986). In *Lect. Notes in Pure and Applied Math.*, volume 125, pages 109–232. New-York, 1990.

[23] F. Chyzak, A. Goyer, and M. Mezzarobba. Symbolic-numeric factorization of differential operators. Technical Report `https://hal.inria.fr/hal-03580658v1`, HAL, 2022.

[24] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. ACM STOC '71*, pages 151–158. 1971.

[25] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Computat.*, 19:297–301, 1965.

[26] J. Doliskani, P. Giorgi, R. Lebreton, and É. Schost. Simultaneous conversions with the Residue Number System using linear algebra. *Transactions on Mathematical Software*, 44(3), 2018. Article 27.

[27] C. Durvye. *Algorithmes pour la décomposition primaire des idéaux polynomiaux de dimension nulle donnés en évaluation.* PhD thesis, Univ. de Versailles (France), 2008.

[28] J. C. Faugère, P. Gianni, D. Lazard, and T. Mora. Efficient computation of zero-dimensional Gröbner bases by change of ordering. *JSC*, 16(4):329–344, 1993.

[29] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1–3):61–88, 1999.

[30] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In T. Mora, editor, *Proc. ISSAC '02*, pages 75–83. Lille, France, July 2002.

[31] J.-C. Faugère, P. Gaudry, L. Huot, and G. Renault. Sub-cubic change of ordering for Gröbner basis: a probabilistic approach. In *Proc. ISSAC '14*, pages 170–177. Kobe, Japan, July 2014.

[32] C. M. Fiduccia. Polynomial evaluation via the division algorithm: the fast Fourier transform revisited. In A. L. Rosenberg, editor, *Proc. ACM STOC '72*, pages 88–93. 1972.

[33] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. FOCM '99*, pages 285–297. 1999.

[34] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. ISSAC 2014*, pages 296–303. Kobe, Japan, July 2014.

[35] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 3rd edition, 2013.

[36] M. Giesbrecht, Q.-L. Huang, and É Schost. Sparse multiplication of multivariate linear differential operators. In *Proc. ISSAC '21*, pages 155–162. New York, USA, 2021.

[37] M. Giusti, K. Hägele, J. Heintz, J. E. Morais, J. L. Montaña, and L. M. Pardo. Lower bounds for diophantine approximation. *Journal of Pure and Applied Algebra*, 117–118:277–317, 1997.

[38] M. Giusti, J. Heintz, J. E. Morais, and L. M. Pardo. When polynomial equation systems can be “solved” fast? In G. Cohen, M. Giusti, and T. Mora, editors, *Proc. AAECC'11*, volume 948 of *Lecture Notes in Computer Science*. Springer Verlag, 1995.

[39] T. Granlund et al. GMP, the GNU multiple precision arithmetic library. `http://www.swox.com/gmp`, 1991.

[40] B. Grenet, J. van der Hoeven, and G. Lecerf. Randomized root finding over finite fields using tangent Graeffe transforms. In *Proc. ISSAC '15*, pages 197–204. New York, NY, USA, 2015. ACM.

[41] G. Hanrot, M. Quercia, and P. Zimmermann. The middle product algorithm I. speeding up the division and square root of power series. *AAECC*, 14:415–438, 2004.

[42] G. Hanrot and P. Zimmermann. A long note on Mulders' short product. *JSC*, 37(3):391–401, 2004.

[43] G. H. Hardy. *Orders of infinity*. Cambridge Univ. Press, 1910.

[44] D. Harvey. Faster exponentials of power series. 2009. `http://arxiv.org/abs/0911.3110`.

[45] D. Harvey. Faster algorithms for the square root and reciprocal of power series. *Math. Comp.*, 80:387–394, 2011.

[46] D. Harvey. Faster truncated integer multiplication. `https://arxiv.org/abs/1703.00640`, 2017.

[47] D. Harvey and J. van der Hoeven. Faster polynomial multiplication over finite fields using cyclotomic coefficient rings. *J. of Complexity*, 54, 2019. Article ID 101404, 18 pages.

[48] D. Harvey and J. van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2):563–617, 2021.

[49] D. Harvey and J. van der Hoeven. Polynomial multiplication over finite fields in time $O(n \log n)$. *JACM*, 69(2), 2022. Article 12.

[50] M. van Hoeij. Factoring polynomials and the knapsack problem. *Journal of Number theory*, 95(2):167–189, 2002.

[51] M. van Hoeij and A. Novocin. Gradual sub-lattice reduction and a new complexity for factoring polynomials. In A. López-Ortiz, editor, *LATIN 2010: Theoretical Informatics*, pages 539–553. Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[52] J. van der Hoeven. Lazy multiplication of formal power series. In W. W. Küchlin, editor, *Proc. ISSAC '97*, pages 17–20. Maui, Hawaii, July 1997.

[53] J. van der Hoeven. Fast evaluation of holonomic functions. *TCS*, 210:199–215, 1999.

[54] J. van der Hoeven. Relax, but don't be too lazy. *JSC*, 34:479–542, 2002.

[55] J. van der Hoeven. The truncated Fourier transform and applications. In *Proc. ISSAC 2004*, pages 290–296. Univ. of Cantabria, Santander, Spain, July 2004.

[56] J. van der Hoeven. Fast composition of numeric power series. Technical Report 2008-09, Université Paris-Sud, Orsay, France, 2008.

[57] J. van der Hoeven. Newton's method and FFT trading. *JSC*, 45(8):857–878, 2010.

[58] J. van der Hoeven. *Journées Nationales de Calcul Formel (2011)*, volume 2 of *Les cours du CIRM*, chapter Calcul analytique. CEDRAM, 2011. Exp. No. 4, 85 pages, `http://ccirm.cedram.org/ccirm-bin/fitem?id=CCIRM_2011__2_1_A4_0`.

[59] J. van der Hoeven. Faster relaxed multiplication. In *Proc. ISSAC '14*, pages 405–412. Kobe, Japan, July 2014.

[60] J. van der Hoeven. On the complexity of polynomial reduction. In *Proc. Applications of Computer Algebra 2015*, volume 198 of *Springer Proceedings in Mathematics and Statistics*, pages 447–458. Cham, 2015. Springer.

[61] J. van der Hoeven. Faster Chinese remaindering. Technical Report, HAL, 2016. `https://hal.archives-ouvertes.fr/hal-01403810`.

[62] J. van der Hoeven. On the complexity of skew arithmetic. *AAECC*, 27(2):105–122, 2016.

[63] J. van der Hoeven. From implicit to recursive equations. *AAECC*, 30(3):243–262, 2018.

[64] J. van der Hoeven. Probably faster multiplication of sparse polynomials. Technical Report, HAL, 2020. `https://hal.archives-ouvertes.fr/hal-02473830`.

[65] J. van der Hoeven and R. Larrieu. Fast Gröbner basis computation and polynomial reduction for generic bivariate ideals. *AAECC*, 30(6):509–539, 2019.

[66] J. van der Hoeven, R. Lebreton, and É. Schost. Structured FFT and TFT: symmetric and lattice polynomials. In *Proc. ISSAC '13*, pages 355–362. Boston, USA, June 2013.

[67] J. van der Hoeven and G. Lecerf. Modular composition via factorization. *J. of Complexity*, 48:36–68, 2018.

[68] J. van der Hoeven and G. Lecerf. Accelerated tower arithmetic. *J. of Complexity*, 55, 2019. Article ID 101402, 26 pages.

[69] J. van der Hoeven and G. Lecerf. Sparse polynomial interpolation. Exploring fast heuristic algorithms over finite fields. Technical Report, HAL, 2019. `https://hal.archives-ouvertes.fr/hal-02382117`.

[70] J. van der Hoeven and G. Lecerf. Fast multivariate multi-point evaluation revisited. *J. of Complexity*, 56, 2020. Article ID 101405, 38 pages.

[71] J. van der Hoeven and G. Lecerf. Ultimate complexity for numerical algorithms. *ACM Commun. Comput. Algebra*, 54(1):1–13, 2020.

[72] J. van der Hoeven and G. Lecerf. Amortized bivariate multi-point evaluation. In *Proc. ISSAC '21*, pages 179–185. New York, NY, USA, 2021. ACM.

[73] J. van der Hoeven and G. Lecerf. Amortized multi-point evaluation of multivariate polynomials. Technical Report, HAL, 2021. `https://hal.archives-ouvertes.fr/hal-03503021`.

[74] J. van der Hoeven and G. Lecerf. Fast amortized multi-point evaluation. *J. of Complexity*, 67, 2021. Article ID 101574, 15 pages.

[75] J. van der Hoeven and G. Lecerf. On the complexity exponent of polynomial system solving. *Found. of Comp. Math.*, 21:1–57, 2021.

[76] J. van der Hoeven and G. Lecerf. Univariate polynomial factorization over large finite fields. *AAECC*, 2022. `https://doi.org/10.1007/s00200-021-00536-1`.

[77] J. van der Hoeven, G. Lecerf, B. Mourrain et al. Mathemagix. 2002. `http://www.mathemagix.org`.

[78] J. van der Hoeven and M. Monagan. Computing one billion roots using the tangent Graeffe method. *ACM SIGSAM Commun. Comput. Algebra*, 54(3):65–85, 2021.

[79] X. Huang and V. Y. Pan. Fast rectangular matrix multiplication and applications. *J. of Complexity*, 14(2):257–299, 1998.

[80] F. Johansson. Arb: a C library for ball arithmetic. *ACM Commun. Comput. Algebra*, 47(3/4):166–169, 2014.

[81] E. Kaltofen. Symbolic computation and complexity theory. In *Proc. ASCM '12*, pages 3–7. Beijing, 2012.

[82] E. Kaltofen and V. Shoup. Fast polynomial factorization over high algebraic extensions of finite fields. In *Proc. ISSAC '97*, pages 184–188. New York, NY, USA, 1997. ACM.

[83] A. Karatsuba and J. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7:595–596, 1963.

[84] K. S. Kedlaya and C. Umans. Fast polynomial factorization and modular composition. *SIAM J. Comput.*, 40(6):1767–1802, 2011.

[85] P. Lairez. A deterministic algorithm to compute approximate roots of polynomial systems in polynomial average time. *Found. Comput. Math.*, 17:1265–1292, 2017.

[86] F. Le Gall and F. Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith–Winograd tensor. In A. Czumaj, editor, *Proc. ACM-SIAM SODA '18*, pages 1029–1046. Philadelphia, PA, USA, 2018.

[87] G. Lecerf. *Une alternative aux méthodes de réécriture pour la résolution des systèmes algébriques.* PhD thesis, École polytechnique, 2001.

[88] G. Lecerf and É. Schost. Fast multivariate power series multiplication in characteristic zero. *SADIO Electronic Journal on Informatics and Operations Research*, 5(1):1–10, 2003.

[89] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[90] M. Mezzarobba. Rigorous multiple-precision evaluation of D-Finite functions in SageMath. Technical Report, HAL, 2016. https://hal.archives-ouvertes.fr/hal-01342769.

[91] M. Mezzarobba. *Autour de l'évaluation numérique des fonctions D-finies.* PhD thesis, École polytechnique, Palaiseau, France.

[92] R. T. Moenck and A. Borodin. Fast modular transforms via division. In *Thirteenth annual IEEE symposium on switching and automata theory*, pages 90–96. Univ. Maryland, College Park, Md., 1972.

[93] N. Möller. On Schönhage's algorithm and subquadratic integer gcd computation. *Math. Comp.*, 77(261):589–607, 2008.

[94] M. Moreno Maza. Design and implementation of multi-threaded algorithms in polynomial algebra. In *Proc. ISSAC '21*, pages 15–20. New York, NY, USA, 2021. ACM.

[95] G. Moroz. New data structure for univariate polynomial approximation and applications to root isolation, numerical multipoint evaluation, and other problems. In *Proc. IEEE FOCS '21.* Denver, United States, 2022.

[96] T. Mulders. On short multiplication and division. *AAECC*, 11(1):69–88, 2000.

[97] V. Neiger, J. Rosenkilde, and G. Solomatov. Generic bivariate multipoint evaluation, interpolation and modular composition with precomputation. In *Proc. ISSAC '20*, pages 388–395. New York, NY, USA, 2020. ACM.

[98] A. Novocin, D. Stehlé, and G. Villard. An LLL-Reduction algorithm with quasi-linear time complexity: extended abstract. In *Proc. ACM STOC '11*, pages 403–412. New York, NY, USA, 2011.

[99] H. J. Nussbaumer. *Fast Fourier Transforms and Convolution Algorithms.* Springer-Verlag, 1981.

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

[101] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[102] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

[103] G. Pólya. Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen. *Acta Mathematica*, 68:145–254, 1937.

[104] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes, The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.

[105] R. Prony. Essai expérimental et analytique sur les lois de la dilatabilité des fluides élastiques et sur celles de la force expansive de la vapeur de l'eau et de la vapeur de l'alkool, à différentes températures. *J. de l'École Polytechnique Floréal et Plairial, an III*, 1:24–76, 1795. Cahier 22.

[106] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo. SPIRAL: code generation for DSP transforms. *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, 93(2):232–275, 2005.

[107] D. S. Roche. What can (and can't) we do with sparse polynomials? In *Proc. ISSAC '18*, pages 25–30. New York, NY, USA, 2018. ACM.

[108] T. Roughgarden. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2021.

[109] F. Rouillier. Solving zero-dimensional systems through the rational univariate representation. *AAECC*, 9:433–461, 1999.

[110] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1(2):139–144, 1971.

[111] A. Schönhage. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Informatica*, 7:395–398, 1977.

[112] A. Schönhage. The fundamental theorem of algebra in terms of computational complexity. Technical Report, Math. Inst. Univ. of Tübingen, 1982.

[113] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

[114] V. Shoup. NTL: a library for doing number theory. 1996. www.shoup.net/ntl.

[115] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:352–356, 1969.

[116] A. L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics*, 4(2):714–716, 1963.

[117] V. Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proc. Int. Cong. of Math. 2018*, volume 4, pages 3465–3506. Rio de Janeiro, 2018.

[118] G. Villard. On computing the resultant of generic bivariate polynomials. In *Proc. ISSAC '18*, pages 391–398. 2018.