The Rôle of Benchmarking in Symbolic Computation
(Position Paper)

James H. Davenport
Department of Computer Science, University of Bath, Bath, BA2 7AY, UK
E-mail: J.H.Davenport@bath.ac.uk

Abstract—There is little doubt that, in the minds of most symbolic computation researchers, the ideal paper consists of a problem statement, a new algorithm, a complexity analysis and preferably a few validating examples. There are many such great papers. This paradigm has served computer algebra well for many years, and indeed continues to do so where it is applicable. However, it is much less applicable to sparse problems, where there are many NP-hardness results, or to many problems coming from algebraic geometry, where the worst-case complexity seems to be rare.

We argue that, in these cases, the field should take a leaf out of the practices of the SAT-solving community, and adopt systematic benchmarking, and benchmarking contests, as a way measuring (and stimulating) progress. This would involve a change of culture.

I. INTRODUCTION

Symbolic computation was an early beneficiary [28] of rigorous complexity theory. This led to the paradigm that the ideal paper consists of a problem statement, a new algorithm, a complexity analysis and preferably a few validating examples. There are many such great papers [12], [17], [29].

This worked fairly well for the fundamental algorithms for dense problems, but less well for sparse problems, which are actually the core subject-matter for practical computer algebra systems. When it comes to more advanced algorithms, we often have (fairly frightening) upper bounds, examples that show that these upper bounds are not as absurd as they might seem on at least some cases, but very little understanding of average-case complexity, or, what the practitioner really wants, “typical case” complexity. For other classes of algorithms, such as integration of algebraic or transcendental functions, there has been very little complexity theory.

II. FUNDAMENTAL ALGORITHMS

A. Dense Polynomials

In the case of dense polynomials, complexity theory produces an excellent understanding of the complexity of polynomial addition, multiplication, division, and a good understanding of the complexity of polynomial greatest common divisor (g.c.d.) computation: at least the complexity of straightforward (computation over \( \mathbb{Z} \)) algorithms, and the worst-case complexity of modular algorithms.

The complexity setting (as opposed to the theory!) is relatively straightforward, one has polynomials in \( n \) variables, of degree \( \leq D \) in each variable, and coefficients of length \( l \) (size \( < 2^t \)). Then the input has size \( (D+1)^n(l+1) \) and the output is similarly bounded. More importantly, the output generally\(^1\) attains its bounds, at least for addition, subtraction, multiplication, and exact division.

B. Sparse Polynomials

For simplicity we consider the sparse distributed representation, as in [37] and as implemented in Maple [32], so a polynomial with \( t \) terms is \( \sum_{i=1}^{t} c_i \prod_{j=1}^{n} x_{i,j}^{\alpha_{i,j}} \) with \( 0 < |c_i| < 2^t \) and \( 0 \leq \alpha_{i,j} \leq D \). Even for multiplication we have the fact that the product of two \( t \)-term polynomials \( (t > 1) \) can have anything between\(^2\) \( 3 \) and \( t^2 \) terms, so we may wish to consider output size as well as input size, rather than just considering \( O(t^2) \) as the obvious bound. Here [37] states the following, which he describes as “nearly within reach”.

Open Problem 1: Develop an algorithm to multiply two sparse polynomials \( f, g \in R[x] \) using \( O(t \log D) \) ring and bit operations, where \( t \) is the number of terms in \( f, g \) and \( fg \), and \( D \) is an upper bound on their degree.

C. Division

For division we have the classical example of \( \frac{x^n-1}{x-1} = x^{n-1} + \cdots + 1 \) with \( n \) terms, so it is now essential to consider output size as well as input size. [37] states the following challenge, which however is not “nearly in reach” when \( g \) is sparse — when \( g \) is dense we compute powers of \( x \) modulo \( g \).

Open Problem 2: Given two sparse polynomials \( f, g \in R[x] \), develop an algorithm to compute the quotient and remainder \( q, r \in R[x] \) such that \( f = q g + r \), using \( O(\log D) \) ring and bit operations, where \( t \) is the number of terms in \( f, g \) and \( q, r, \) and \( \deg f < D \).

[19, Challenge 3] shows that even the decision problem “does \( g \) divide \( f \) exactly” is unknown.

Open Problem 3: Either

- find a class of problems for which the problem “does \( g \) divide \( f \)” is NP-complete; or

\(^1\)There are exceptions such as \( f - f \), or multiplications where the coefficients of the output are smaller than those of the inputs, but these are rare.

\(^2\)Consider \( \left[ (x-1) \frac{x^n-2^n}{x-2} \right] \cdot \left[ (x-2) \frac{x^n-1}{x-1} \right] = x^{2n} - (2^n+1)x^n + 2^n \).
• find an algorithm for the divisibility of polynomials which is polynomial-time.

D. Greatest Common Divisors

Again it is necessary to consider output size, as the neat example of [38] shows:
\[ \gcd(x^{pq} - 1, x^{pq} - x^p - x^q + 1) = x^{p+q-1} - x^{p+q-2} \pm \ldots - 1. \]

Most of the classic results in this area are due to Plaisted [34], [35], [36], as in the following result.

**Theorem 1 ([35]):** It is NP-hard to determine whether two sparse polynomials (in the standard encoding) have a non-trivial common divisor.

The basic device of the proofs is to encode the NP-complete problem of 3-satisfiability so that a formula \( W \) in \( n \) Boolean variables goes to a sparse polynomial \( p_M(W) \) which vanishes exactly at certain \( M \)th roots of unity corresponding to the satisfiable assignments to the formula \( W \), where \( M \) is the product of the first \( n \) primes. [MR 85j:68043]

We have previously [19, Challenge 2] posed the following.

**Open Problem 4:** Either

- find a class of problems for which the g.c.d. problem is still NP-complete even when cyclotomic factors are explicitly encoded (see Appendix A); or
- find an algorithm for the g.c.d. of polynomials with no cyclotomic factors, which is polynomial-time in the standard encoding.

As this is undecided, the state of the art seems to be that even the decision problem (output size one bit) for greatest common divisors can be NP-hard on some (probably rare) problems.

This paper proposed the position that the methodology of computer algebra research has not really adapted to the fact that NP-hardness (or worse) seems to be core to much of its actual challenges.

III. MORE ADVANCED PROBLEMS

A. Polynomial Factorization

Practically all known polynomial factorization algorithms begin by doing a square-free decomposition, and this is also hard in theory.

**Theorem 2 ([27]):** Over \( \mathbb{Z} \) and in the standard sparse encoding, the two problems

1) deciding if a polynomial is square-free
2) deciding if two polynomials have a non-trivial g.c.d.

are equivalent under randomized polynomial-time reduction.

Hence, in the light of Theorem 1, determining square-freeness is hard, at least when polynomials with cyclotomic factors are involved.

In the dense univariate case, we know that the worst-case complexity is polynomial in \( n \) [29]. The \( n \) worst-case complexity of polynomial factorization is to the existence of Swinnerton-Dyer polynomials (those that factor compatibly modulo every prime, but are irreducible). Various improvements have been suggested [1], [40], [41], but we lack a systematic comparison. [24] is the nearest we have, but the definitions of his polynomials are referred to the NTL website, which seems not to have them.

Since almost all polynomials are irreducible in the sense that
\[ \lim_{H \to \infty} \frac{|\{\text{such polynomials that factor}\}|}{|\{\text{polynomials of degree } d, \text{ coefficients } \leq H\}|} = 0, \]

typical-case complexity isn’t helpful.

Hence polynomial factorization papers nearly always rely on a set of examples to demonstrate their superiority (e.g. [42] drawing on [16]). Hardware progress (as well as some algorithmic improvements) have made this particular set of problems trivial, and there doesn’t seem to be an agreed corpus of hard multivariate problems. [2] generated large problems at random, but do not seem to have preserved them, so there is little for reproducibility researchers to build on.

B. Gröbner bases

There is a strain of papers, culminating in [30], that shows the computation of a Gröbner base to be worst-case doubly-exponential (in \( n \), the number of indeterminates), as the polynomials must have that degree. The author used to believe that this was caused by the multiple components in the construction, but this belief was punctured by [15] who constructs a prime ideal whose representation has polynomials of doubly-exponential degree.

Nevertheless, most Gröbner base problems, while often difficult, seem not be in this class. Hence there has been interest in the Gröbner community in benchmarking and sets of test problems, which were collected by the POSSO project [7]. However, this was very much a one-off effort, and the collection is not particularly usable (we seem to have lost some of the sources and are forced to re-engineer typeset documents) and many of the problems are now trivial, due to algorithmic improvements (and some hardware progress). Hence the community could really do with a modern equivalent.

C. Regular Chains

The method of triangular decompositions/regular chains has been proposed as an alternative to Gröbner bases. Until relatively recently, less was known about its complexity, but [4] has filled some serious gaps in our knowledge. In particular their complexities are singly exponential in \( n \). It has to be said that the distinction between \( d^{2n^2} \) and \( d^{n^2} \) only manifests itself for \( n > 14 \): currently totally impracticable. It is also not clear how rare the bad cases are for this algorithm either. They may be related to bad cases for Gröbner bases, since both are based on very large outputs being generated, but this is not fully understood (at least by the author!). In the presence of such uncertainties, a library of examples would be useful.

\(^3\) A community effort to reconstruct these would be useful!
It would be particularly useful to consider the performance of both Gröbner methods and Regular Chains methods on the same sets of examples. To the best of the author’s knowledge, this has never been done in any systematic way.

D. Real Geometry

A major algorithm in this area is cylindrical algebraic decomposition, whose cost is doubly-exponential in $n$, and there are quantifier elimination examples whose output size is actually doubly-exponential [11]. However, these require a number of quantifier alternations that is $O(n)$, and this is known to be necessary for doubly-exponential complexity [23]. Of course if one writes down a fully quantified statement at random, the average number of alternations is $O(n)$, but that doesn’t mean that this situation is “typical”, whatever that might mean. Indeed, there are two very different scenarios.

Games Let the opponent’s moves be $O_i$, and my moves $M_j$. Then the first question “is situation $S$ immediately fatal” is

$$\neg \forall O_1 \exists M_1 : \text{playable}(S/O_1/M_1).$$

The next question is

$$\neg \forall O_1 \exists M_1 \forall O_2 \exists M_2 : \text{playable}(S/O_1/M_1/O_2/M_2),$$

and so on.

SMT

The basic Satisfiability Modulo Theories problem is purely existential. Variants on it have been proposed [31], but these tend to have a fixed alternation structure, and a more complex problem has more quantified variables at each level, rather than more levels.

In the presence of very bad worst-case complexity, and a belief that “typical” examples are much better, but exhibit varied characteristics, some in this field have also resorted to collecting examples, e.g. [43]. A more recent set of examples, [33], is deposited in a formal data sharing repository.

A recent paper in this field [10] does use some of the benchmarking descriptive techniques borrowed from the SAT/SMT field and described in Section IV.

E. Weak Complexity

An idea that originally appeared in [3] is that of weak complexity, where the statement $f(n) \in O(g(n))$ holds outside a set whose measure tends exponentially to 0 as $n \to \infty$. This captures the idea of their being “only a few” bad examples, but that they might be so bad that a straight average would still be dominated by them. In [13] this was applied to the computation of the homology groups of the closed semi-algebraic set defined by a Boolean combination of $=, \leq, \geq$, so falls in the ambit of Section III-D.

The requirement “tends exponentially to 0” is a strong one, stronger than, for example “almost all polynomials are irreducible” [8].

F. Real Geometry etc.

In the areas of symbolic integration, summation and o.d.e. solving, very little is usually written about the complexity: essentially because the input language is too rich to provide any useful statements. Instead it is usual to rely on collections such as [26].

IV. BENCHMARKING METHODOLOGY IN SAT/SMT

The fields of Boolean Satisfiability (SAT) and its derivative Satisfiability Modulo Theories (SMT) have been faced with NP-completeness (or worse for SMT) since their inception. Hence they have resorted to systematic benchmarking and annual contests. Rather than the list of 10–15 polynomials found in [16], [42], these contests include thousands of problems. Many of these come from actual examples, others are deliberately contrived to be difficult [39]. The winner is then, at that time, the best single state-of-the-art solver. [44] introduced the concept of the virtual best solver (VBS): a hypothetical solver that uses the best existing solver for that problem on each problem. If the VBS does much better than any individual solver, one can then ask whether it is possible to build a portfolio solver that attempts to mimic the VBS. Some progress here is discussed in [21]. However, much larger datasets are required for machine learning to build a portfolio system than symbolic computation generally has [25], and the difficulties in getting such datasets are described in [22].

However, if one has thousands of benchmark examples, there is little point in publishing a table of respective performances on each problem, as is traditionally done in symbolic computation. Instead, various graphical techniques are used, as described in [9]. An example is given in Figure 1, where it can be seen that:

1) Z3 is ultimately the best solver.
2) But Colibri solves more problems in a short time (< 1 second) than any other solver.
3) VBS is significantly better than any individual solver, both in terms of number of problems solved and time, so there is substantial room for a portfolio approach.

V. DIRECTIONS?

Though the SAT community has been benchmarking for far longer, their problems have little syntactic variety. The author feels that Computer Algebra should rather look at the SMT Community, where there are a range of domain-specific contests under a common umbrella: see http://smtcomp.sourceforge.net/2018/. However, it is quite possible that there are other role models of which the author is unaware. The following requirements seems unavoidable if computer algebra is to run these sorts of competitions.

1) A common input language. The SAT community finds this easy with the DIMACS standard. The SMT community has evolved, and is continuing to evolve, the SMT-LIB [6] language. Algebra in general has the OpenMath standard [14]. The relationship between SMT-LIB and OpenMath is discussed in [20].
2) A shared repository. This is now much easier with tools like SourceForge or GitHub than it was in the days of [7].
3) A (probably rotating) set of competition organisers.
4) A position in the subject’s calendar (for SMT it is at the annual SMT workshop, for computer algebra

---

Footnotes:

[1] The researchers may well wish to analyse such a table in private, of course.

it could be at ISSAC, or another annual conferences: the author made such a call at ACA 2018 [18]).

There are also challenges.

- Load time — SAT solvers minimise this, but computer algebra historically hasn’t cared. Furthermore, operations such as Gröbner bases tend to be “load on demand”. It is not clear what the right answer is here.

- Benchmarking in the presence of garbage collection. Some SMT solvers also garbage collect, and running in a fixed memory size seems to answer this problem.

- The contest runs on fixed servers. Many people in computer algebra seem to use laptops, but repeatable timing here is challenging [5].

- There is also the question of parallelism and multicore architecture. The SMT competition specifically specifies that the servers will be four-core machines. It could be argued that this sort of achine is standard now, and that systems that can’t take advantage of it deserve to be penalised. More pragmatically, it is not clear what else to do.

ACKNOWLEDGMENTS

The author is grateful to Martin Brain, Matthew England, Zak Tonks and the SYNASC reviewers for their comments, though the views expressed here are not necessarily anyone else’s. This work was supported by EU H2020-FETOPEN-2016-2017-CSA project SC² (712689).

REFERENCES


A. Cyclotomics

Many of the known hard examples, or reductions to NP-hard problems, come from cyclotomic polynomials. Hence we might consider explicitly representing them in one of the encodings $C_n(x) = x^n - 1$ or $\Phi_n(x) = \prod_{k, \gcd(k, n) = 1} \left( x - \exp\left( \frac{2\pi ik}{n} \right) \right)$.

These are related by the following result.

**Proposition 1**: $C_n(x) = \prod_{d|n, \mu(n/d) = -1} \Phi_d(x)$ and $\Phi_n(x) = \prod_{d|n, \mu(n/d) = 1} C_d(x)^{\mu(n/d)}$, where $\mu$ is the Möbius function.

This was suggested in [19] but little progress has been made since. It is worth noting that we need to handle shifted cyclotomics, as in $2^n C_n(x^2) = x^{2n} - 2^n$. However, it is not necessary to consider $x^{2n} - 2^n$, since polynomials of this form do not seem to produce similar special cases. $x^{2mn} - 2^m$ would need to be viewed as $2^m C_{2m}(x^2)$. 

APPENDIX

A. Cyclotomics

Many of the known hard examples, or reductions to NP-hard problems, come from cyclotomic polynomials. Hence we might consider explicitly representing them in one of the encodings $C_n(x) = x^n - 1$ or $\Phi_n(x) = \prod_{k, \gcd(k, n) = 1} \left( x - \exp\left( \frac{2\pi ik}{n} \right) \right)$.

These are related by the following result.

**Proposition 1**: $C_n(x) = \prod_{d|n, \mu(n/d) = -1} \Phi_d(x)$ and $\Phi_n(x) = \prod_{d|n, \mu(n/d) = 1} C_d(x)^{\mu(n/d)}$, where $\mu$ is the Möbius function.

This was suggested in [19] but little progress has been made since. It is worth noting that we need to handle shifted cyclotomics, as in $2^n C_n(x^2) = x^{2n} - 2^n$. However, it is not necessary to consider $x^{2n} - 2^n$, since polynomials of this form do not seem to produce similar special cases. $x^{2mn} - 2^m$ would need to be viewed as $2^m C_{2m}(x^2)$. 

APPENDIX

A. Cyclotomics

Many of the known hard examples, or reductions to NP-hard problems, come from cyclotomic polynomials. Hence we might consider explicitly representing them in one of the encodings $C_n(x) = x^n - 1$ or $\Phi_n(x) = \prod_{k, \gcd(k, n) = 1} \left( x - \exp\left( \frac{2\pi ik}{n} \right) \right)$.

These are related by the following result.

**Proposition 1**: $C_n(x) = \prod_{d|n, \mu(n/d) = -1} \Phi_d(x)$ and $\Phi_n(x) = \prod_{d|n, \mu(n/d) = 1} C_d(x)^{\mu(n/d)}$, where $\mu$ is the Möbius function.

This was suggested in [19] but little progress has been made since. It is worth noting that we need to handle shifted cyclotomics, as in $2^n C_n(x^2) = x^{2n} - 2^n$. However, it is not necessary to consider $x^{2n} - 2^n$, since polynomials of this form do not seem to produce similar special cases. $x^{2mn} - 2^m$ would need to be viewed as $2^m C_{2m}(x^2)$.