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 [24] 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 [10], [14], [25].

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 algebaric or transcendental, 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 $\mathbf{Z}$) 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 $< 2^l$). Then the input has size $n(D+1)(l+1)$ 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 [32] and as implemented in Maple [27], so a polynomial with t terms is $\sum_{i=1}^{t} c_i \prod_{j=1}^{n} x_j^{\alpha_{i,j}}$ with $0 < |c_i| < 2^l$ 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 4 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 lower bound. Here [32] 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 $\widetilde{O}(t \log D)$ 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. [32] states the following challenge, which however is not "nearly in reach" when g is sparse — when g is dense we compute power 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 $f = qg + r$, using $\widetilde{O}(t \log D)$ ring and bit operations, where t is the number of terms in f, g *and* q and r, and $\deg f < D$.

[16, 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
- find an algorithm for the divisibility of polynomials which is polynomial-time.

[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.

D. Greatest Common Divisors

Again it is necessary to consider output size, as the neat example of [33] shows:

$$\gcd(x^{pq}-1, x^{p+q}-x^p-x^q+1) = x^{p+q-1}-x^{p+q-2}\pm\cdots-1.$$

Most of the classic results in this are are due to Plaisted [29], [30], [31], as in the following result.

Theorem 1 ([30]): 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 Mth 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 [16, 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 ([23]): Over $\mathbf{Z}$ 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.

Even in the dense case, very little is known about the worst-case complexity of polynomial factorization, due to the existence of Swinnerton-Dyer polynomials (those that factor compatibly modulo every prime, but are irreducible). Since almost all polynomials are irreducible in the sense that $\forall d > 0$

$$\lim_{H\to\infty} \frac{|\{\text{such polynomials that factor}\}|}{|\{\text{polynomials of degree } d\text{, coefficients} \le H\}|} = 0, \quad (1)$$

typical-case complexity isn't helpful.

Hence polynomial factorization papers nearly always rely on a set of examples to demonstrate their superiority (e.g. [35] drawing on [13]). 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 problems.

B. Gröbner bases

There is a strain of papers, culminating in [26], 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 [12] 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 field in benchmarking and sets of test problems, which were collected by the POSSO project [5]. 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[2]) 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 [2] 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^{2^n} and d^{5n^3} 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!).

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 side is actually doubly-exponential [9]. However, these require a number of quantifier *alternations* that is $O(n)$, and this is known to be necessary for doubly-exponential complexity [20]. Of course if one writes down a fully quantified statement at random, the average number of alternations is $O(n)$, but that

[2] A community effort to reconstruct these would be useful!

doesn't mean that this situation is "typical", whatever that might mean.

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. [36]. A more recent set of examples, [28], is deposited in a formal data sharing repository[3].

A recent paper in this field [8] 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 [1] 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 [11] 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" [6].

F. Integration etc.

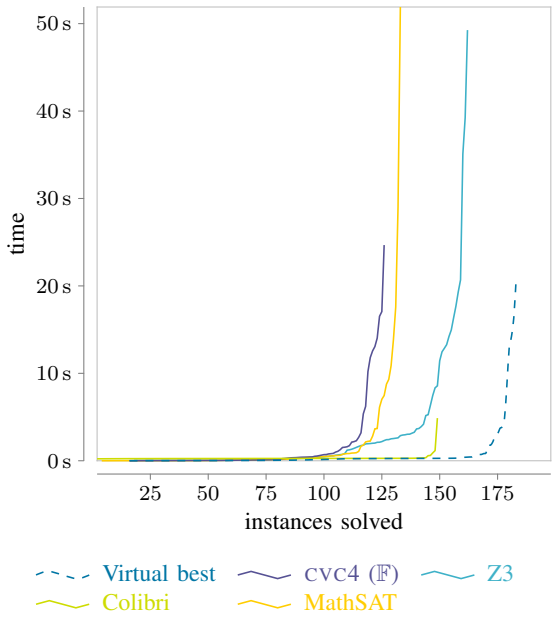
In the areas of symbolic integration, summation and o.d.e. solving, very little is usually written about the complexity: essentially because the input language is too rich to provide any useful statements. Instead it is usual to rely on collections such as [22].

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 [13], [35], these contests include thousands of problems. Many of these come from actual examples, others are deliberately contrived to be difficult [34]. The winner is then, at that time, the best single state-of-the-art solver. [37] 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 [18]. However, much larger datasets are required for machine learning to build a portfolio system than symbolic computation generally has [21], and the difficulties in getting such datasets are described in [19].

However, if one has thousands of benchmark examples, there is little point in publishing[4] a table of respective performances on each problem, as is traditionally done in symbolic computation. Instead, various graphical techniques are used, as described in [7]. An example is given in Figure 1, where it can be seen that:

Fig. 1: [7, Figure 12], with legend moved



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 corts of competitions.

1) A common input language [17].
2) A shared repository. This is now much easier with tools like SourceForge than it was in the days of [5].
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 it could be at ISSAC, or another annual conferences: the author made such a call at ACA 2018 [15]).

There are also challenges.

- Load time — SAT solvers minimise this, and computer algebra historically hasn't cared.

[3] https://doi.org/10.5281/zenodo.1226892, and with an encoding in a widespread benchmark format SMT2 [4].

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

- Benchmarking in the presence of garbage collection. Some SMT solvers also garbage collect, and running in a fixed memory size seems to answer this.
- The cotest runs on fixed servers. Many people in computer algebra seem to use laptops, but repeatable timing here is challenging [3]

ACKNOWLEDGMENTS

The author is grateful to Martin Brain, Matthew England and Zak Tonks 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 $\mathcal{SC}^2$ (712689).

REFERENCES

[1] D. Amelunxen and M. Lotz. Average-case complexity without the black swans. *J. Complexity*, 41:82–101, 2017.

[2] E. Amzallag, G. Pogudin, M. Sun, and N.T. Vo. Complexity of Triangular Representations of Algebraic Sets. https://arxiv.org/abs/1609.09824v6, 2018.

[3] Bharathan Balaji, John McCullough, Rajesh K Gupta, and Yuvraj Agarwal. Accurate characterization of the variability in power consumption in modern mobile processors. In *Hotpower 12*, pages 29:1–29:5, 2012.

[4] C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.6 (draft 5 June 2017). http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-draft-2017-06-05.pdf, 2017.

[5] D. Bini and B. Mourrain. Polynomial test suite. http://www-sop.inria.fr/saga/POL/, 1996.

[6] C. Borst, E. Boyd, C. Brekken, S. Solberg, M.M. Wood, and P.M. Wood. Irreducibility of random polynomials. *To appear in Experimental Mathematics*, pages 1–9.

[7] M.N. Brain, J.H. Davenport, and A. Griggio. Benchmarking Solvers, SAT-style. *SC^2 2017 Satisfiability Checking and Symbolic Computation CEUR Workshop*, 1974(RP3):1–15, 2017.

[8] C.W. Brown. Projection and Quantifier Elimination Using Non-uniform Cylindrical Algebraic Decomposition. In *Proceedings ISSAC 2017*, pages 53–60, 2017.

[9] C.W. Brown and J.H. Davenport. The Complexity of Quantifier Elimination and Cylindrical Algebraic Decomposition. In C.W. Brown, editor, *Proceedings ISSAC 2007*, pages 54–60, 2007.

[10] W.S. Brown. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors. *J. ACM*, 18:478–504, 1971.

[11] P. Bürgisser, F. Cucker, and J. Tonelli-Cueto. Computing the Homology of Semialgebraic Sets I: Lax Formulas. https://arxiv.org/abs/1807.06435, 2018.

[12] A.L. Chistov. Double-exponential lower bound for the degree of any system of generators of a polynomial prime ideal. *St. Petersburg Math. J.*, 20:983–1001, 2009.

[13] B.G. Claybrook. Factorization of multivariate polynomials over the integers. *SIGSAM Bulletin*, 10:13–13, 1976.

[14] G.E. Collins. Subresultants and Reduced Polynomial Remainder Sequences. *J. ACM*, 14:128–142, 1967.

[15] J.H. Davenport. Lessons between Computer Algebra and Verification/Satisfiability Checking (Presentation at ACA 2018). http://staff.bath.ac.uk/masjhd/Slides/ACA2018-JHD.pdf, 2018.

[16] J.H. Davenport and J. Carette. The Sparsity Challenges. In S. Watt *et al.*, editor, *Proceedings SYNASC 2009*, pages 3–7, 2010.

[17] J.H. Davenport, M. England, R. Sebastiani, and P. Trentin. OpenMath and SMT-LIB. http://arxiv.org/abs/1803.01592, 2018.

[18] M. England. Machine Learning for Mathematical Software. In J.H. Davenport, M. Kauers, G. Labahn, and J. Urban, editors, *Proceedings Mathematical Software — ICMS 2018*, pages 165–174, 2018.

[19] M. England and J.H. Davenport. Experience with Heuristics, Benchmarks & Standards for Cylindrical Algebraic Decomposition. *International Workshop on Satisfiability Checking and Symbolic Computation 2016 CEUR WS*, 1804:24–31, 2017.

[20] D.Yu. Grigoriev and N.N. Vorobjov Jr. Solving Systems of Polynomial Inequalities in Subexponential Time. *J. Symbolic Comp.*, 5:37–64, 1988.

[21] Z. Huang, M. England, D. Wilson, J.H. Davenport, L.C. Paulson, and J. Bridge. Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. In S.M.Watt *et al.*, editor, *Proceedings CICM 2014*, pages 92–107, 2014.

[22] E. Kamke. Differential Gleichungen-Lösungsmethoden und Lösungen. *Chelsea*, 1959.

[23] M. Karpinski and I. Shparlinski. On the Computational Hardness of Testing Square-Freeness of Sparse Polynomials. In M. Fossorier, H. Imai, S. Lin, and A. Poli, editors, *Proceedings AAECC-13*, pages 492–497, 1999.

[24] D.E. Knuth. *The Art of Computer Programming, Vol. II, Seminumerical Algorithms*. Addison-Wesley, 1969.

[25] A.K. Lenstra, H.W. Lenstra Jun., and L. Lovász. Factoring Polynomials with Rational Coefficients. *Math. Ann.*, 261:515–534, 1982.

[26] E.W. Mayr and S. Ritscher. Dimension-dependent bounds for Gröbner bases of polynomial ideals. *J. Symbolic Comp.*, 49:78–94, 2013.

[27] M. Monagan and R. Pearce. POLY : A new polynomial data structure for Maple 17. In R. Feng *et al.*, editor, *Proceedings Computer Mathematics*, pages 325–348, 2014.

[28] C.B. Mulligan, R. Bradford, J.H. Davenport, M. England, and Z. Tonks. Non-linear Real Arithmetic Benchmarks derived from Automated Reasoning in Economics. *Proc. SC-Square 2018 ceur-ws.org*, 2189:48–60, 2018.

[29] D.A. Plaisted. Sparse Complex Polynomials and Irreducibility. *J. Comp. Syst. Sci.*, 14:210–221, 1977.

[30] D.A. Plaisted. Some Polynomial and Integer Divisibility Problems are NP-Hard. *SIAM J. Comp.*, 7:458–464, 1978.

[31] D.A. Plaisted. New NP-Hard and NP-Complete Polynomial and Integer Divisibility Problems. *Theor. Comp. Sci.*, 31:125–138, 1984.

[32] D.S. Roche. What Can (and Can't) we Do with Sparse Polynomials? In *Proceedings ISSAC 2018*, pages 25–30, 2018.

[33] A. Schinzel. On the greatest common divisor of two univariate polynomials, I. In *A Panorama of number theory or the view from Baker's garden*, pages 337–352. C.U.P., 2003.

[34] I. Spence. Weakening Cardinality Constraints Creates Harder Satisfiability Benchmarks. *J. Exp. Algorithmics Article 1.4*, 20, 2015.

[35] P.S. Wang. An Improved Multivariable Polynomial Factorising Algorithm. *Math. Comp.*, 32:1215–1231, 1978.

[36] D.J. Wilson, R.J. Bradford, and J.H. Davenport. A Repository for CAD Examples. *ACM Comm. Computer Algebra 3*, 46:67–69, 2012.

[37] L. Xu, F. Hutter, H.H. Hoos, and K. Leyton-Brown. Evaluating component solver contributions to portfolio-based algorithm selectors. *Theory Appl. Satisfiability Testing SAT 2012*, pages 228–241, 2012.

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 $\Phi_n(x) = \prod_{\substack{k=1 \\ \gcd(k,n)=1}}^{n} \left(x - e^{2\pi i k/n}\right)$. These are related by the following result.

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

This was suggested in [16] but little progress has been made since. It is worth noting that we need to handle shifted cyclotomics, as in $2^n C_n(\frac{x}{2}) = x^n - 2^n$. However, it is not necessary to consider $x^n - 2$, since polynomials of this form do not seem to produce similar special cases. $x^{mn} - 2^m$ would need to be viewed as $2^m C_m(\frac{x^n}{2})$.