Fast Algorithms and Libraries for Polynomials

NSERC Discovery Grant 2019–2024

I work in Computer Algebra or Symbolic Computation. Computer Algebra Systems, like Magma, Maple, Mathematica, Singular and Sage, are used by scientists and engineers in industry and academia. They enable many kinds of non-numerical mathematical calculations to be undertaken. One core facility they provide is computation with polynomials in one or more variables over various coefficient rings. This is the main area I propose to work in.

I am designing algebraic algorithms and implementing them in Maple and in C. Since 2009, when multi-core computers were becoming mainstream, and Computer Algebra Systems needed to exploit them, I started to also develop and implement parallel algorithms. I am using Cilk C for multi-core implementations.

My **long term objectives** are (1) to build a solid foundation for our Computer Algebra Systems by designing good algorithms for multivariate polynomial multiplication, division, GCD, and factorization and (2) to undertake high-performance parallel implementations of them and make the software available to users and the research community, either by integrating it into Maple or by making a standalone C/C^{++} library available on the web.

I propose the following four **short-term objectives**. Each is central to Computer Algebra and has many applications. Each is suitable for graduate student involvement.

P1. Design and implement a first high-performance polynomial time algorithm for factoring multivariate polynomials with integer coefficients.

P2. Apply sparse interpolation tools to compute greatest common divisors (GCDs) of multivariate polynomials with algebraic number and algebraic function coefficients.

P3. Develop and apply sparse interpolation tools to compute the Dixon resultant, which is used to eliminate variables from systems of polynomial equations.

P4. Build a C^{++} library for multivariate polynomials that supports modular algorithms, has tools for sparse polynomial interpolation, and provides multi-core and AVX support.

The main tool connecting research problems P1, P2, P3 and P4 is **sparse polynomial interpolation**. Let me review this and explain why it is important.

A polynomial $f(x_1, \ldots, x_n)$ with $\deg(f, x_i) < d$ may have up to d^n terms. We say f is sparse if it has relatively few terms, less than $\sqrt{d^n}$ terms say. Let t be the number of terms of f. The main idea is to interpolate f from values modulo a prime p. We want algorithms whose complexity depends on the actual number of terms t and not on the possible number d^n. The main reason this is important is that in most applications, polynomials with many variables are sparse and often very sparse.

Zippel's probabilistic algorithm from [33] was the first sparse polynomial interpolation method. It interpolates f using $O(ndt)$ points. The Ben-Or/Tiwari algorithm from [1] interpolates f using $2t$ points but it works over $\mathbb{Q}$ and one needs to know t.

A mod p version of Ben-Or/Tiwari was first developed by Kaltofen, Lakshman and Wiley in [16]. A different mod p version of Ben-Or/Tiwari is presented by Murao and Fujise in [28]. It allows one to use a smaller prime p. In [17], Kaltofen, Lee and Lobo show how to determine t with high probability using $2t + 2$ points.

For projects P2 and P3 we need to also interpolate rational functions such as $(2t_1^3 - 3t_2t_3 + t_4^3)/(t_1^4 + t_2^2t_3^2 + t_4^4)$. Kaltofen and Trager in [15] gave the first sparse rational function interpolation algorithm. For k parameters, the best result I know of is Cuyt and Lee [4], which uses a factor of $O(k)$ fewer images than my RATZIP algorithm from [20].

The literature on sparse interpolation algorithms is substantial. See Daniel Roche's 2018 paper "What we can (and can't do) with sparse polynomials" in [29], and the practical work of van der Hoven and Lecerf [11]. But few researchers are applying sparse interpolation to applications, and, as a tool, it is not getting deployed in Computer Algebra Systems. My research proposal addresses this. Details of the 4 projects follow.

P1 High-performance multivariate polynomial factorization.

All Computer Algebra Systems basically use Wang's "variable at a time, degree at a time" Hensel lifting from [31] to factor multivariate polynomials. At each step one solves a polynomial Diophantine equation using the same "variable at a time, degree at a time" strategy. This is highly sequential and it can be exponential in the number of variables n. Polynomial time solutions (see Zippel [34] and Kaltofen [14]) are known but are not practical.

In [25] my PhD student Tuncer and I presented a new polynomial time algorithm which exploits the structure between the coefficients of the factors we are trying to find. Our implementation beats the implementation of Wang's algorithm in Maple, Magma and Singular on a wide range of inputs, so it is practical. What I am even more excited about is that our structural observation opens the door to a fast parallel design.

Our approach in [25] used sparse interpolation to solve the multivariate Diophantine equations that arise in Hensel lifting. The new idea I want to develop is to Hensel lift bivariate images of the factors (in parallel) and interpolate the coefficients of the factors (in parallel) from these bivariate images. I made a first experiment in Cilk C this Spring for polynomials with two factors which I presented at ICMS 2018 [26]. I am training my new PhD student Tian Chen to help me develop this approach to handle multiple factors, improve the performance, and analyze the failure probability of the algorithm.

This strategy reduces Hensel lifting in $\mathbb{Z}_p[x_1, ..., x_n]$ to (i) multi-point polynomial evaluation, (ii) many Hensel lifts in $\mathbb{Z}_p[x_1, x_2]$ of modest degree, and (iii) solving Vandermonde linear systems. In trying to speed (ii) up, I have discovered a beautiful cubic algorithm for bivariate Hensel lifting in $\mathbb{Z}_p[x_1, x_2]$.

To lift two factors of degree n in x and degree m in y, linear and quadratic Hensel lifting both do $O(m^2n^2)$ arithmetic operations in $\mathbb{Z}_p$ if classical quadratic polynomial arithmetic is used, [2]. My cubic algorithm does $O(mn^2 + m^2n)$. Although not as fast asymptotically as fast quadratic Hensel lifting which is linear in mn up to logarithmic factors [7], my C code for my cubic algorithm is much faster and uses much less space than Magma's fast quadratic Hensel lifting up to m=n=10000 which is as far as I could go before Magma exceeded the memory (64 gigabytes) on my machine.

I have worked out how to make the cubic algorithm also work for Hensel lifting in $\mathbb{Z}[x_1]$. But does it handle multiple factors and non-monic factors? If yes, then it could become the algorithm of choice in practice for factoring polynomials in $\mathbb{Z}[x_1]$ and $\mathbb{Z}_p[x_1, x_2]$. My Masters' student Garrett Paluck will continue to investigate this.

P2 Computing polynomial GCDs over number fields and function fields.

Computer Algebra Systems often have to simplify fractions A/B where A and B are polynomials. To do this they compute and divide out by $G = \mathrm{GCD}(A, B)$. GCDs arise in many other places as well. They are perhaps the most important operation for overall efficiency of a Computer Algebra System because computing a polynomial GCD is much more expensive than adding, multiplying and dividing polynomials.

For polynomials with integer coefficients, many Computer Algebra Systems use Zippel's sparse GCD algorithm from [33]. For A and B in $n+1$ variables $x_0, x_1, \ldots, x_n$, Zippel's algorithm interpolates $G = \mathrm{GCD}(A, B) = \sum_{i=0}^{d_0} c_i(x_1, \ldots, x_n) x_0^i$ using $O(t(d_1 + \cdots + d_n))$ monic images in x_0 where $d_i = \deg(G, x_i)$ and t is the maximum number of terms of the c_i. In [12, 13], my PhD student Hu and I designed a new sparse GCD algorithm that needs only $2t + 2$ images. It uses a Kronecker substitution on $x_1, \ldots, x_n$. I want to try this approach for polynomials with algebraic number and algebraic function coefficients.

Algebraic numbers like $\sqrt{3}$ and algebraic functions like $\sqrt{1-t^2}$ arise naturally in geometric problems and in applications involving polynomial equations. Without a good modular GCD algorithm, the default Euclidean algorithm will often not terminate.

Given polynomials A, B in $K[x_1, \ldots, x_n]$ where $K = \mathbb{Q}(\alpha_1, ..., \alpha_k)$ is a number field, or a function field in variables $t_1, \ldots, t_r$, a basic approach is to compute $G = \mathrm{GCD}(A, B)$ modulo primes, combine them with Chinese remaindering, then use rational number reconstruction. This approach was developed by Encarnacion [5] for $\mathbb{Q}(\alpha_1)[x_1]$, extended to $\mathbb{Q}(\alpha_1, \ldots, \alpha_k)[x_1]$ by van Hoeij and me in [8] and subsequently generalized by us in [9] to the algebraic function case using dense rational function interpolation.

To compute G modulo p for the number field case (and function field case respectively), I propose to interpolate the variables (and parameters) in G from monic images of the form

$$x_1^d + \mathbb{Z}_p[\alpha_1, \ldots, \alpha_k][x_1] \quad \text{and} \quad x_1^d + \mathbb{Z}_p(t_1)[\alpha_1, \ldots, \alpha_k][x_1]$$

respectively. I will try using a Kronecker substitution on $x_2, \ldots, x_n$ and a second Kronecker substitution on $t_2, \ldots, t_r$. One technical difficulty is the possibility of hitting a zero divisor in the finite ring $\mathbb{Z}_p[\alpha_1, ..., \alpha_k]$ at some point in the algorithm. This may be due to the choice of prime, evaluation point or Kronecker substitution.

To bound the probability of hitting a zero divisor we will need degree bounds for G in x_i and t_j and integer coefficient bounds for G where G is a factor of A and B. This is likely difficult for $k > 1$ thus more suited to PhD students than Masters students. For $k > 1$ one approach would be to use a primitive element γ to map the input number field $\mathbb{Q}(\alpha_1, \ldots, \alpha_k)$ into $\mathbb{Q}(\gamma)$ where bounds will be easier to get. Also, because there are three types of variables, x_i, α_j and t_k, an implementation will be complicated. For this, my Maple package `RECDEN` that I developed and used in [8] will be helpful.

P3 Computing the Dixon resultant.

Given a system of polynomial equations in $(x_1, ..., x_n)$, one way to solve the system is to first eliminate $n-1$ variables to obtain a polynomial R in one variable, say x_1. The polynomial R is called a resultant or eliminant. In many applications there are also parameters $y_1, ..., y_k$ present, for example, lengths. So R is a polynomial in $k+1$ variables $x_1, y_1, ..., y_k$.

There are three general approaches that can be used to eliminate variables: (1) Gröbner Bases, (2) Triangular Sets, and (3) Determinants. I am interested in the Dixon matrix A. The determinant of A is called the Dixon resultant. In general, it a multiple of R. Among the determinant methods, it is preferred because the Dixon matrix A is smaller and the Dixon resultant has fewer spurious factors – see Kapur and Saxema [19].

In [18], Kapur, Saxema and Lakshman observed that the Dixon resultant is often 0, thus providing no useful information. They showed that if one picks any submatrix M of A of maximum rank then R is a factor of $D = \det(M)$. They gave an effective method for computing D and found that this was much faster than computing a Gröbner basis.

In [22, 23, 23], Lewis lists a large number of elimination problems with parameters from real applications where Gröbner basis engines, including Magma's and FGb [6], fail. They typically run out of memory after several hours, whereas computing $D = \det M$ takes a few minutes. I have tried Maple's Triangularize command from the `RegularChains` library which was developed by Moreno Maza et. al. [21, 27]. It computes a Triangular Set. It also fails on Lewis' problems.

To compute $D = \det M$, Lewis modified Gaussian elimination to keep track of any factors of D that appear in the elimination. But working in the fraction field $\mathbb{Q}(x_1, y_1, \ldots, y_k)$ results in an expression swell. On some large examples, Lewis had to manually choose a different minor of A to get his code to terminate.

I propose to use sparse interpolation to directly interpolate D. Actually, Kapur and Saxena already tried this in [19]. They used Zippel's sparse interpolation which requires $O(kdt)$ evaluation points where k is the number of parameters, t is the number of terms in D and $d = \deg(D)$. A direct application of our modified Ben-Or/Tiwari method from [12] requires only $2t + 2$ points, so it should be much faster. However, the root finding in the sparse interpolation does not parallelize – this is an open problem in Computer Algebra. Instead, I propose to interpolate D from univariate images in x_1 or, alternatively, from bivariate images in x_1, y_1. This will likely reduce t, hence reduce the sparse interpolation cost, and inject some parallelism into the sparse interpolation.

In principle, sparse interpolation techniques can also be applied to approaches (1) Gröbner Bases and (2) Triangular Sets. For these one would interpolate $\texttt{monic}(R)$ in $\mathbb{Q}(y_1, \ldots, y_k)[x_1]$ from monic univariate images in x_1. Since the coefficients of $\texttt{monic}(R)$ are in $\mathbb{Q}(y_1, \ldots, y_n)$ this approach requires that we either interpolate rational functions or we scale by a multiple of the LCM of the denominators of $\texttt{monic}(R)$ so that we can use sparse polynomial interpolation which is cheaper. One source of such a multiple is the leading coefficient of $\det M$ in x_1.

When trying to reproduce Lewis' results, I discovered something very interesting about the Dixon matrix M. On half of Lewis' examples, M has a non-trivial block structure, that is, $\det(M) = \det(B_1)\det(B_2)\cdots\det(B_m)$ where the B_k are nontrivial blocks. For example, one of Lewis' problems has eight blocks of size 48, 48, 49, 49, 50, 50, 52, 53. Obviously this makes computing $\det(M)$ much easier. Furthermore R is often a factor of each $\det(B_k)$.

Why does the Dixon matrix have this block structure? Can we identify the smallest block B of M with R a factor of $\det B$? How fast can we interpolate $\det B$? Why do Gröbner bases and Triangular sets fail so badly on Lewis' problems? Does sparse interpolation "fix" the failure of Gröbner bases and Triangular set methods?

P4 A high-performance library for multivariate polynomials.

I propose to develop a C^{++} library for multivariate polynomials with multi-core and AVX support. The library will provide tools for sparse polynomial and rational function interpolation and other tools needed to implement modular algorithms including a mixed radix representation for long integers. I will use the 128 bit integer type in the Gnu C compiler to support 127 bit primes as well as 63 bit primes so that sparse interpolation methods can handle problems with more variables.

I know of two research groups that are developing software libraries for multivariate polynomials, both in C^{++}, but with different goals than mine. Moreno Maza [3] and his students are developing the BPAS (Basic Polynomial Algebra Subroutines) library. It is aimed at computing Triangular Sets. They use FFT based techniques and CilkPlus for multicore processors. J. van der Hoven, Lecerf and Mourrain are developing the Mathemagix library [10]. It is aimed at asymptotically fast methods.

I have designed and implemented in C, algorithms and data structures for multivariate polynomials. I have been a major contributor of code to the Maple kernel. I plan to continue to design and implement parallel algorithms for multi-core computers using Cilk C/C^{++} but also start using Intel's SIMD technology AVX. Intel's newest AVX processor, the AVX-512, can do eight 64 bit arithmetic operations at a time. Actually, it can do 16 "Fused Multiply Adds" at a time. How and where do we exploit this?

One bottleneck in project P1 is multi-point polynomial evaluation modulo a prime p. Given a polynomial $A \in \mathbb{Z}_p[x_1, x_2, \ldots, x_n]$ in expanded form with a million terms, we want to evaluate $A(x_1, x_2, \beta_2^j, \ldots, \beta_n^j) \bmod p$ for $j = 1, 2, \ldots, 2t+2$. I have started a collaboration with Pierre Fortin (Sorbonne, France) to apply AVX technology to this problem.

This project is suitable for students interested in HPC tools and their application.

Impact: The new approach I am using in project P1 will likely lead to a first high-performance multivariate polynomial factorization code. P2 is a necessary evil. No one is working on polynomial GCDs with algebraic function field coefficients because it is difficult. Yet, without a good solution, inputs with many variables will cause Computer Algebra Systems to fail to terminate. P3 will potentially lead to new fast parallel algorithms for eliminating variables from polynomial systems with parameters. P4 is something I would like to do for the research community. Sparse polynomial and rational function interpolation is complicated; we need a good open source library for it so others can deploy it.

Methodology: As my students and I design algorithms for P1–P3, we will be making experimental prototype implementations of them. Implementations will enable us to see design problems clearly and force us to work through details. For this I will use Maple or Magma because coding in them is much easier than coding in C and C^{++}. Since the algorithms in P1–P3 are probabilistic, we will need to determine worst case failure probabilities. For this I will use tools from algebra and combinatorics like the Schwartz-Zippel Lemma [30]. For (Cilk) C and C^{++} implementations, my students can use my own serial C library for polynomials. We will identify (parallel) bottlenecks in the code and try to speed those up. This may lead to a new research problem or suggest a change in the algorithm. For Project P4 I will first build a library of serial codes for sparse polynomial and rational function interpolation before adding parallel routines and AVX support. We will always benchmark our software with existing software.

References

[1] Michael Ben-Or and Prasson Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. *Proceedings of STOC '20*, pp. 301–309, ACM, 1988.

[2] Laurent Bernardin. On Bivariate Hensel Lifting and its Parallelization. *Proceedings of ISSAC '98*, pp. 96–100, ACM, 1998.

[3] Changbo Chen, Syatoslav Covanov, Farnam Mansouri, Marc Moreno Maza, Ning Xie, and Yuzhen Xie. The Basic Polynomial Algebra Subprograms. *Proceedings of ICMS 2014*, pp. 669–676, Springer, 2014. See also www.bpaslib.org for the BPAS website.

[4] Annie Cuyt and Wen-shin Lee. Sparse interpolation of multivariate rational functions. *J. Theoretical Comp. Sci.* **412**:1445–1456, Elsevier, 2011.

[5] Mark Encarnacion. Computing GCDs of polynomials over algebraic number fields. *J. Symbolic Computation*, **20**: 299-313, 1995.

[6] Jean-Charles Faugère, FGb. https://www-polsys.lip6.fr/ jcf/FGb/index.html

[7] von zur Gathen and Gerhard. *Modern Computer Algebra*, Cambridge, 3rd ed., 2013.

[8] Mark van Hoeij and Michael Monagan. A modular GCD algorithm over number fields presented with multiple field extensions. *Proceedings of ISSAC 2002*, pp. 109-116, ACM, 2002.

[9] M. van Hoeij, M. B. Monagan. Algorithms for Polynomial GCD Computation over Algebraic Function Fields. *Proceedings of ISSAC '2004*, pp. 297–304, ACM, 2004.

[10] van der Hoven, Lecerf and Mourrain. Mathemagix. www.mathemagix.org

[11] Joris van der Hoven and Gregoire Lecerf. Sparse polynomial interpolation in practice. *Communications in Computer Algebra*, **48**:187–191, ACM, 2014.

[12] Jiaxiong Hu and Michael Monagan. A fast parallel sparse polynomial GCD algorithm. *Proceedings of ISSAC 2016*, pp. 271–278, ACM, 2016.

[13] Jiaxiong Hu and Michael Monagan. A fast parallel sparse polynomial GCD algorithm. Submitted April 2018 to *J. Symbolic Computation*, (40 pages).

[14] Kaltofen, E., Sparse Hensel lifting. *Proceedings of EUROCAL '85*, LNCS **204**:4–17, Springer, 1985.

[15] Eric Kaltofen and Barry Trager. Computing with polynomials given by black boxes for their evaluations. Greatest common divisors, factorization, separation of numerators and denominators. *J. Symbolic Comput.*, **9**(3):301-320, 1990.

[16] Erich Katofen, Lakshman Y.N., John-Michael Wiley. Modular Rational Sparse Multivariate Polynomial Interpolation. *Proceedings of ISSAC 1990*, pp. 135–139, ACM, 1990.

[17] Erich Kaltofen, Wen-shin Lee and Austin Lobo. Early Termination in Ben-Or/Tiwari Sparse Interpolation and a Hybrid of Zippel's Algorithm. *Proceedings of ISSAC 2000*, pp. 192–201, ACM, 2000.

[18] Deepak Kapur, Tushar Saxena and Lu Yang. Algebraic and Geometric Reasoning using Dixon Resultants. *Proceedings of ISSAC '94*, pp. 99–107, ACM, 1994.

[19] Deepak Kapur and Tusha Saxena. Comparison of various multivariate resultant formulations. *Proceedings of ISSAC '95*, pp. 187–194, ACM, 1995.

[20] Jennifer de Kleine, Michael Monagan and Allan Wittkopf. Algorithms for the Non-monic case of the Sparse Modular GCD Algorithm. *Proceedings of ISSAC '2005*, pp. 124–131, ACM, 2005.

[21] F. Lemaire, M. Moreno Maza, Y. Xie. The RegularChains Library in Maple. *Maple Conference 2005 Proceedings*, pp. 355–368, Maplesoft, 2005.

[22] Robert Lewis. Dixon-EDF: The Premier Method for Solution of Parametric Polynomial Systems. *Applications of Computer Algebra 2015*, Proceedings in Mathematics and Statistics, **198**, pp. 238–256, Springer, 2017.

[23] Robert Lewis. Resultants, Implicit Parameterizations, and Intersections of Surfaces. *Proceedings of ICMS 2018*, LNCS 10931 pp. 310–318, Springer, 2018.

[24] Robert Lewis. Image Analysis: Identification of Objects via Polynomial Systems, *Proceedings of ICMS 2018*, LNCS 10931, pp. 305–309, Springer 2018.

[25] Michael Monagan and Baris Tuncer. Using Sparse Interpolation in Hensel Lifting. *Proceedings of CASC 2016*, Springer-Verlag LNCS **9890**, pp. 381–400, 2016.

[26] Michael Monagan and Baris Tuncer. Sparse multivariate Hensel lifting: A high-performance design and implementation. *Proceedings of ICMS 2018*, LNCS **10931**, pp. 359-368, Springer, 2018.

[27] Marc Moreno Maza. The Regular Chains Library. http://www.regularchains.org/

[28] Hirokazu Murao and Tetsuro Fujise. Modular Algorithm for Sparse Multivariate Polynomial Interpolation and its Parallel Implementation. *J. Symbolic Cmpt.* **21**:377–396, 1996.

[29] Daniel Roche. What we Can (and Can't) do with Sparse Polynomials. *Proceedings of ISSAC 2018*, pp. 25–30, ACM, 2018.

[30] The Schwartz-Zippel Lemma. https://en.wikipedia.org/wiki/Schwartz

[31] Paul S. Wang. An improved Multivariate Polynomial Factoring Algorithm. *Mathematics of Computation*, **32**(144):1215–1231, 1978.

[32] Applications of Resultants. https://en.wikipedia.org/wiki/Resultant

[33] Richard Zippel. Probabilistic algorithms for sparse polynomials. *Proceedings of EUROSAM '79*, pp. 216–226. Springer-Verlag, 1979.

[34] Richard Zippel. Newton's iteration and the sparse Hensel algorithm. *Proceedings of SYMSAC '81*, pp. 68–72, ACM, 1981.