ISSAC 2009 Poster Abstracts

Communicated by I. S. Kotsireas, A. Novocin, P. Horn

Probabilistic Algorithms for Polynomial Absolute Factorization

C. Bertone[a,b] G. Chèze[c] A. Galligo[a]

[a]Laboratoire J.-A. Dieudonné, Université de Nice - Sophia Antipolis, Parc Valrose, 06108 Nice Cedex 02, France

[b] Dipartimento di Matematica, Università degli Studi di Torino, via Carlo Alberto 10, Torino, Italy.

[c]Institut de Mathématiques de Toulouse, Université Paul Sabatier Toulouse 3, MIP Bât 1R3, 31 062 Toulouse cedex 9, France.

In the last years many authors contributed to develop and implement algorithms on polynomial factorization and even if the situation evolved rapidly, there is still room for improvements and new points of view. We focus on absolute factorization of rationally irreducible polynomials with integer coefficients. For such polynomials, the best current algorithm and implementation is Chèze's ([1], [2]) presented at Issac'04; it is based on semi-numerical computation, uses LLL and is implemented in Magma; it can factorize polynomials of high degrees, up to 200. One of the challenges is to improve its capabilities at least in certain situations.

We propose yet another strategy and algorithm to deal with bivariate absolute irreducibility test and factorization, but the techniques extend to the multivariate case. The poster will present a simple, but very efficient, Las Vegas irreducibility test for an integer polynomial $f(X,Y)$ irreducible over $\mathbb{Q}$, based on a property of the Newton polygon of f. We also extend the strategy to get a factorization algorithm based on modular computations.

Starting from a factorization of $f(X,Y)$ modulo a "well-chosen" prime p, we can obtain the algebraic field extension $\mathbb{Q}(\alpha)$ in which the absolute factors of $f(X,Y)$ live.
We can assume that α is an integer linear combination of the coefficients of an absolute factor of $f(X,Y)$, more precisely we define $\alpha := f_1(x_0, y_0)$, with $f_1(X,Y)$ irreducible absolute factor and $x_0, y_0 \in \mathbb{Z}$.
First we estimate a sufficient level of accuracy for a p-adic approximation of the algebraic integer α and then lift the "mod$p$" factorization of $f(x_0, Y)$ through Hensel liftings (to the computed accuracy level).
Finally we construct an integer polynomial of the right degree vanishing on α through the algorithm LLL applied to a matrix involving the p-adic approximation and the level of accuracy.

Our absolute factorization algorithm can be viewed as a drastic improvement of the classical algorithm TKTD (see [3], [4], [5]) as we replace the computations in an algebraic extension of $\mathbb{Q}$ of degree n, the degree of the input polynomial, by computations in an extension of the minimal degree s, the number of absolute factors of the input polynomial.

We made a preliminary implementation in Maple and computed several examples. It is very promising as it is fast and able to compute the researched algebraic extension for high degree polynomials. Some examples can be found on http://math.unice.fr/∼cbertone/ .

For instance, we constructed, using two random polynomials and a resultant, a rational irreducible polynomial $f(X,Y)$ of degree 200 with 10 absolute factors of degree 20 each, with $\|f(X,Y)\|_\infty \sim 10^{70}$.
It took less than 60 seconds, using a non-optimal implementation in Maple 10, to define the algebraic extension in which the absolute factors live. More precisely:

- We factorized $f(X,Y) \mod 47$ in 33 seconds.
- We then estimated the level of accuracy needed to construct the algebraic extension, obtaining 898. It was sufficient to use Hensel Lifting on the factorization until the level 47^{512}, and this took 2.8 seconds.
- Finally, we found a polynomial $q(T) \in \mathbb{Q}[T]$ defining the algebraic field extension using LLL in 14 seconds.

The bottleneck of the procedure is now the final x-adic Hensel lifting.

In other words, we believe that our approach will allow that the practical complexity of absolute factorization of polynomials with integer coefficients decreases to the practical complexity of rational polynomial factorization.

References

[1] G. Chèze. Absolute polynomial factorization in several variables and the knapsack problem. In *Proceedings of ISSAC 2004*: 87–94, 2004.

[2] G. Chèze. Des méthodes symboliques-numériques et exactes pour la factorisation absolue des polynomes en deux variables. PhD thesis, Univ. Nice, France, 2004.

[3] R. Dvornicich, C. Traverso, Newton symmetric functions and the arithmetic of algebraically closed fields. Applied algebra, algebraic algorithms and error-correcting codes (Menorca, 1987), *Lecture Notes in Comput. Sci.*, 356: 216–224, 1989.

[4] E. Kaltofen. Fast parallel absolute irreducibility testing. *J. Symbolic Comput.*, 1(1): 57–67, 1985.

[5] B. Trager. On the integration of algebraic functions. PhD Thesis, 1985.

Applying Linear Algebra Routines to Modular Ore Polynomial Matrix Algorithms

Howard Cheng[a] **George Labahn**[b]

[a] Dept. of Mathematics and Computer Science University of Lethbridge, Lethbridge, Canada
e-mail: howard.cheng@uleth.ca
[b] Symbolic Computation Group, School of Computer Science, University of Waterloo, Waterloo, Canada
e-mail: glabahn@uwaterloo.ca

Ore polynomials provide a general setting for describing linear differential, difference, and q-difference operators. Systems of equations defined by these operators can be represented by matrices of Ore polynomials. The FFreduce algorithm [1] performs row operations in a fraction-free way to transform such matrices into "simpler" ones while controlling coefficient growth. This algorithm can be used to compute the row-reduced and weak Popov forms of shift polynomial matrices [1], as well as the Popov form of general Ore polynomial matrices [3]. It can also be used to compute a greatest common right divisor (GCRD) and a least common left multiple (LCLM) of such matrices.

A modular version of the FFreduce algorithm was developed by the authors to reduce the computational complexity [2]. In the modular algorithm, it was observed that the evaluation reduction $\mathbb{Z}_p[t][Z;\sigma,\delta] \to \mathbb{Z}_p[Z;\sigma,\delta]$ is not generally an Ore ring homomorphism [5]. Instead of performing the row operations on the Ore polynomial matrices directly, larger striped Krylov matrices over $\mathbb{Z}_p$ was constructed and row reductions were performed on these matrices. Each Krylov matrix was constructed dynamically—rows were added depending on which row is selected as the pivot in each step. Unlucky homomorphisms corresponds to divisors of the determinant of a certain Krylov matrix, and they occur rarely.

In practice, the resulting modular algorithm was only slightly faster than the corresponding fraction-free algorithm for very large inputs. One obstacle in obtaining further improvement was that the row operations to reduce the

Krylov matrix have to be done one step at a time, because it is not possible to predict which Krylov matrix should be constructed a priori. As a result, only low-level linear algebra subroutines in the `LinearAlgebra:-Modular` package was used to accelerate the computation.

In this work, we investigate the applicability of higher-level linear algebra subroutines to speed up the computation. Assuming that the first evaluation point is "lucky," the Krylov matrices for the remaining evaluation points can be constructed and the entire matrix can be reduced with a few calls to the appropriate linear algebra subroutines. This should allow more sophisticated implementations to speed up the reduction process (e.g. [4]).

References

[1] B. Beckermann, H. Cheng, and G. Labahn. Fraction-free row reduction of matrices of Ore polynomials. *Journal of Symbolic Computation*, 41(5):513–543, 2006.

[2] H. Cheng and G. Labahn. Modular computation for matrices of Ore polynomials. In *Computer Algebra 2006: Latest Advances in Symbolic Algorithms*, pages 43–66, 2007.

[3] P. Davies, H. Cheng, and G. Labahn. Computing Popov form of general Ore polynomial matrices. In *Milestones in Computer Algebra (MICA) 2008*, pages 149–156, 2008.

[4] J.-G. Dumas, P. Giorgi, and C. Pernet. FFPACK: finite field linear algebra package. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, pages 119–126. ACM, 2004.

[5] Z. Li and I. Nemes. A modular algorithm for computing greatest common right divisors of ore polynomials. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, pages 282–289. ACM, 1997.

Symbolic-Numeric Sparse Interpolation of Multivariate Rational Functions

Annie Cuyt[a] Wen-shin Lee[a]

[a] Department of Mathematics and Computer Science, University of Antwerp,Middelheimlaan 1, 2020 Antwerpen, Belgium

{annie.cuyt,wen-shin.lee}@ua.ac.be

http://www.win.ua.ac.be/~cant

http://www.wen-shin.com

Consider the problem of sparse interpolation for a black-box multivariate rational function

$$f(x_1,\ldots,x_n)=\frac{u(x_1,\ldots,x_n)}{v(x_1,\ldots,x_n)}, \tag{1}$$

in floating point arithmetic, in which both the numerator and denominator

$$\begin{aligned} u(x_1,\ldots,x_n)&=\sum_{j=1}^{\ell} a_j x_1^{d_{j_1}}\cdots x_n^{d_{j_n}}, \quad a_j\neq 0,\\ v(x_1,\ldots,x_n)&=\sum_{k=1}^{m} b_k x_1^{e_{k_1}}\cdots x_n^{e_{k_n}}, \quad b_k\neq 0 \end{aligned} \tag{2}$$

are polynomials with complex coefficients, $u(x_1,\ldots,x_n), v(x_1,\ldots,x_n)\in\mathbb{C}[x_1,\ldots,x_n]$.

That is, to recover coefficients a_j, b_k and multivariate exponents $(d_{j_1}, \ldots, d_{j_n})$, $(e_{k_1}, \ldots, e_{k_n})$ for $1 \le j \le \ell$ and $1 \le j \le m$ in (2) from black-box evaluations of (1) in a finite precision environment.

We present a symbolic-numeric interpolation method that is sensitive to the sparsity of the black-box multivariate rational function. Our method implements the homogenization from [1, 3] and numerically interpolates the modified rational function with respect to the newly introduced homogenizing variable. Then by combining with the numerical sparse polynomial interpolation from [2], we simultaneously recover the multivariate exponents of non-zero terms in both $u(x_1, \ldots, x_n)$ and $v(x_1, \ldots, x_n)$. Once all such non-zero terms are recovered, various techniques can be utilized to determine the corresponding coefficients a_j and b_k, and the given multivariate rational function can be interpolated.

For example, let

$$f(x_1, \ldots, x_n) = \frac{a_1 x_1^{d_{1\,1}} \cdots x_n^{d_{1\,n}} + \cdots + a_\ell x_1^{d_{\ell\,1}} \cdots x_n^{d_{\ell\,n}}}{1 + b_2 x_1^{e_{2\,1}} \cdots x_n^{e_{2\,n}} + \cdots + b_m x_1^{e_{m\,1}} \cdots x_n^{e_{m\,n}}}$$

be defined at $(0, \ldots, 0)$. By introducing the homogenizing variable z, we obtain a modified rational function $F(z, x_1, \ldots, x_n) = f(x_1 z, \ldots, x_n z)$.

Suppose $p_1, \ldots, p_n \in \mathbb{Z}_{\ge 0}$ are pairwise relatively prime. We fix $(x_1, \ldots, x_n)$ at $(p_1, \ldots, p_n)$ and consider the univariate rational interpolation of $F(z, p_1, \ldots, p_n)$ with respect to z,

$$F(z, p_1, \ldots, p_n) = \frac{A_1(p_1, \ldots, p_n) z^{\delta_1} + \cdots + A_\lambda(p_1, \ldots, p_n) z^{\delta_\lambda}}{1 + B_2(p_1, \ldots, p_n) z^{\epsilon_2} + \cdots + B_\mu(p_1, \ldots, p_n) z^{\epsilon_\mu}}. \tag{3}$$

In (3), the coefficients $A_1(p_1, \ldots, p_n)$, $\ldots$, $A_\lambda(p_1, \ldots, p_n)$, $B_2(p_1, \ldots, p_n)$, $\ldots$, $B_\mu(p_1, \ldots, p_n)$ are multivariate polynomials $A_1(x_1, \ldots, x_n)$, $\ldots$, $A_\lambda(x_1, \ldots, x_n)$, $B_2(x_1, \ldots, x_n)$, $\ldots$, $B_\mu(x_1, \ldots, x_n)$ evaluated at $(p_1, \ldots, p_n)$.

We continue to interpolate the rational functions $F(z, p_1^2, \ldots, p_n^2)$, $F(z, p_1^3, \ldots, p_n^3)$, $\ldots$. From each interpolation, we obtain a set of coefficients that are the corresponding coefficient polynomials evaluated at powers: $A_1(p_1^2, \ldots, p_n^2)$, $\ldots$, $B_\mu(p_1^2, \ldots, p_n^2)$, $A_1(p_1^3, \ldots, p_n^3)$, $\ldots$, $B_\mu(p_1^3, \ldots, p_n^3)$, $\ldots$.

We take a look at the interpolation of polynomial A_1. Using the numerical sparse polynomial interpolation from [2], we can interpolate $A_1(x_1, \ldots, x_n)$ from $A_1(p_1, \ldots, p_n)$, $A_1(p_1^2, \ldots, p_n^2)$, $A_1(p_1^3, \ldots, p_n^3)$, $\ldots$.

For $i = 1, 2, \ldots$, from the coefficients of each interpolated $F(z, p_1^i, \ldots, p_n^i)$ we can simultaneously interpolate coefficient polynomials $A_1, \ldots, A_\lambda, B_2, \ldots, B_\mu$ by the numerical sparse polynomial interpolation [2]. When all the coefficient polynomials $A_1, \ldots, A_\lambda, B_2, \ldots, B_\mu$ are recovered, the original rational function $f(x_1, \ldots, x_n)$ can be reconstructed.

Our rational interpolation is an iterative method and does not require the knowledge on either the degrees or number of terms in the black-box rational function. The number of evaluations and interpolation steps depend on the number of non-zero terms in the given rational function and the accuracy required.

We also investigate other sparse rational interpolation approaches and relevant issues, including the situation when the given function $f(x_1, \ldots, x_n)$ is not defined at the origin $(0, \ldots, 0)$. Some initial tests are demonstrated in Maple.

References

[1] A. Díaz, E. Kaltofen, FOXBOX: a system for manipulating symbolic objects in black box representation, in: Proc. 1998 Internat. Symp. Symbolic Algebraic Comput. (ISSAC 1998), pages 30–37, ACM, 1998.

[2] A. Cuyt, W.-s. Lee, A new algorithm for sparse interpolation of multivariate polynomials, Theoretical Computer Science, 409(2), pages 180–185, 2008.

[3] E. Kaltofen, W.-s. Lee, A. A. Lobo, Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm, in: Proc. 2000 Internat. Symp. Symbolic Algebraic Comput. (ISSAC 2000), pages 192–201, ACM, 2000.

On least fixed points of systems of positive polynomials

Javier Esparza[a] **Andreas Gaiser**[a] **Stefan Kiefer**[a]

[a] Faculty of Computer Science, Technische Universität München, Germany
esparza@in.tum.de,gaiser@model.in.tum.de,kiefer@in.tum.de

We present an algorithm to compute the least nonnegative solution of a *system of probabilistic polynomials (SPrP)*, a system of equations of the form

$$X_1 = f_1(X_1, ..., X_n) \quad ... \quad X_n = f_n(X_1, ..., X_n)$$

where, for every $i \in \{1, ..., n\}$, f_i is a polynomial over $X_1, \ldots, X_n$ with positive rational coefficients that *add up to 1*. The solutions of a SPrP are the fixed points of the function $f : \mathbb{R}^n \to \mathbb{R}^n$ with $f = (f_1, ..., f_n)$. For example, $X_1 = \frac{1}{2}X_1X_2 + \frac{1}{2}, X_2 = \frac{1}{4}X_2X_2 + \frac{1}{4}X_1 + \frac{1}{2}$ is a SPrP with $f(X_1, X_2) = (\frac{1}{2}X_1X_2 + \frac{1}{2}, \frac{1}{4}X_2X_2 + \frac{1}{4}X_1 + \frac{1}{2})$. Obviously, $\overline{1} = (1, ..., 1)$ is a solution for every SPrP. By Kleene's theorem, every SPrP has a least nonnegative solution (called just least solution in what follows).

SPrPs are important in different areas of the theory of stochastic processes and computational models. A fundamental result of the theory of branching processes, with numerous applications in physics and biology (see e.g. [6, 1]), states that extinction probabilities of species are equal to the least solution of a SPrP. The same result has been recently shown for the probability of termination of certain probabilistic recursive programs ([5, 4]). The consistency of stochastic context-free grammars, a problem of interest in statistical natural language processing, also reduces to checking whether the least solution of a SPrP equals $\overline{1}$ (see e.g. [8]).

We fix an SPrP with function f and denote its least solution by μ_f. The following two problems are motivated by the applications above: (1) decide whether $\mu_f = \overline{1}$, and (2) given $\epsilon > 0$ compute lb, ub such that $lb \leq \mu_f \leq ub$ and $ub - lb \leq \overline{\epsilon}$.

Etessami and Yannakakis show in [5] that Problem (1) can be solved in polynomial time using exact Linear Programming; however, it is known that in practice this method is inefficient (over 30 minutes in average for Maple's exact Simplex on random examples with 200 variables). The same experiments show that inexact Linear Programming solvers based on floating point arithmetic easily lead to false results because of severe numerical problems (10 variables and coefficients from the set $\{0.01, 0.5, 0.49\}$ are enough to break lpsolve). Concerning Problem (2), lower bounds for μ_f can be computed using Newton's method to approximate a root of the function $f(\overline{X}) - \overline{X}$ (the roots obviously correspond to the fixed points of f). It is shown in [5] and [3, 7] that if Newton's method starts at a *prefixed point* p of f (a point $p \in [0, 1]^n$ such that $f(p) \geq p$), for example $p = \overline{0}$, and exact arithmetic is used, then the method converges to μ_f and every approximant is again a prefixed point. Again, experiments show that naively using exact arithmetic is very inefficient. In [9] a tool is presented that uses Newton's method with floating point arithmetic, but for the same 7-variable example as above the tool wrongly indicates that $\mu_f = \overline{1}$. Finally, the computation of *upper* bounds for μ_f does not seem to have been considered so far.

The poster presents our new algorithm based on Newton's method to simultaneously solve Problems (1) and (2). To the best of our knowledge, it is the first algorithm usable in practice that never returns a wrong result and provides not only a lower but also an upper bound for the solution.

The algorithm, shown in Figure 1, handles *normalized* SPrPs where f_i is of quadratical degree and the Jacobian matrix $f'(\overline{1})$ is an irreducible matrix such that $Id - f'(\overline{1})$ is nonsingular. It is essentially sufficient to consider this class of SPrPs, since we can transform an arbitrary SPrP into a collection of SPrPs of this form, although here some problems still have to be solved. The algorithm is based on the following theorem, which builds on a result of [5] and Perron-Frobenius theory. (By "$\prec$" resp. "$\succ$" we denote smaller resp. greater in every component.)

Theorem 1. *If f is normalized and $\mu_f = \overline{1}$, then there is an $i \in \mathbb{N}$ such that $f'(\overline{1})(\overline{1} - \nu^{(i)}) \prec (\overline{1} - \nu^{(i)})$ where $\nu^{(i)}$ is the i-th Newton iterant. If $\mu_f < \overline{1}$, then the same property holds with "$\prec$" replaced by "$\succ$".*

We have proved that, with exact arithmetic, the index i has polynomial size in the size of the SPrP. The proof relies on recent results about the convergence speed of Newton's method [3, 7] and on a gap theorem by Dedieu [2].

To overcome the inefficiency of exact arithmetic, we compute each Newton iterant using inexact arithmetic of variable precision, and then check whether it is a prefixed point. If so, rounding errors do not affect correctness, and the computation can proceed. If not, the computation of the iterant is repeated with increased precision; in our prototype implementation we increment the bitsize of floating point numbers. In order to obtain upper bounds on μ_f, if our method detects that $\mu_f \neq \overline{1}$, we compute a *postfixed point*, i.e. a $p \in [0,1]^n$ with $p \geq f(p)$, which can be used as a first upper bound for μ_f. We can show that the sequence $p, f(p), f(f(p)), \ldots$ converges to μ_f from above with linear convergence rate.

$\nu^{(0)} \Leftarrow \overline{0}$
$i \Leftarrow 0$
while true **do**
  $\nu^{(i+1)} \Leftarrow$ `NewtonNumeric`$(f, \nu^{(i)})$
  if $\nu^{(i+1)}$ no prefixed point or $\nu^{(i+1)} \not\succ \nu^{(i)}$ **then**
    $\nu^{(i+1)} \Leftarrow \nu^{(i)}$
    increase precision of `NewtonNumeric`
  else if $f'(\overline{1}) \cdot (\overline{1} - \nu^{(i+1)}) \succ \overline{1} - \nu^{(i+1)}$ **then**
    calculate postfixed point p using $\nu^{(i+1)}$
    compute $\nu^{(i+1+j)}$ and $f^k(p)$ for increasing j, k
      until $f^k(p) - \nu^{(i+1+j)} \leq \overline{\epsilon}$
    return $(\nu^{(i+1+j)}, f^k(p))$
  else if $f'(\overline{1}) \cdot (\overline{1} - \nu^{(i+1)}) \prec \overline{1} - \nu^{(i+1)}$ **then**
    return $(\overline{1}, \overline{1})$
  else
    $i \Leftarrow i + 1$
  end if
end while

Figure 1: The algorithm returns (lb, ub) with $lb \leq \mu_f \leq ub$.

In order to compute the elements of the sequence we again use inexact arithmetic and increase precision when needed. In this way we obtain a sequence of increasingly better upper bounds for μ_f which, together with the lower bounds provided by the Newton iterants, yield a solution to Problem (2).

The price to pay for a correctness guarantee with inexact arithmetic is the possibility that the algorithm does not terminate. This can only happen when $\mu_f = \overline{1}$. While in our experiments the algorithm always terminates, we are currently searching for easy conditions that guarantee termination. Our claim that ours is the first algorithm usable in practice that never returns a wrong result is supported by a speed-up factor of 15 on random systems with 200 variables over Maple's exact Simplex. Recall also that Linear Programming only solves problem (1), while our algorithm simultaneously solves (1) and (2).

References

[1] Krishna B. Athreya and Peter E. Ney. *Branching Processes*. Springer-Verlag, 1972.

[2] Jean-Pierre Dedieu. Estimations for the separation number of a polynomial system. *Journal of Symbolic Computation*, 24(6):683 – 693, 1997.

[3] Javier Esparza, Stefan Kiefer, and Michael Luttenberger. Convergence thresholds of Newton's method for monotone polynomial equations. In *Proceedings of the 25th International Symposium on Theoretical Aspects of Computer Science*, pages 289–300, 2008.

[4] Javier Esparza, Antonín Kučera, and Richard Mayr. Model checking probabilistic pushdown automata. In *LICS 2004*. IEEE Computer Society, 2004.

[5] Kousha Etessami and Mikhalis Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1):1–66, 2009.

[6] Theodore E. Harris. *The theory of branching processes*. Springer-Verlag, Berlin, 1963.

[7] Stefan Kiefer, Michael Luttenberger, and Javier Esparza. On the convergence of Newton's method for monotone systems of polynomial equations. In *Proceedings of the 39th ACM Symposium on Theory of Computing*, pages 217–226. ACM, 2007.

[8] Christopher D. Manning and Hinrich Schuetze. *Foundations of Statistical Natural Language Processing*. The MIT Press, June 1999.

[9] Dominik Wojtczak and Kousha Etessami. Premo : An analyzer for probabilistic recursive models. In *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 66–71. Springer, 2007.

Implementation of Boolean Gröbner Bases in Risa/Asir

Shutaro Inoue[a] **Yosuke Sato**[a]

[a] Tokyo University of Science
e-mail: inoue@mi.kagu.tus.ac.jp
e-mail: ysato@rs.kagu.tus.ac.jp

A commutative ring $\mathbf{B}$ with an identity is called a *boolean ring* if every element of which is idempotent. A residue class ring $\mathbf{B}[X_1, \ldots, X_n]/\langle X_1^2 - X_1, \ldots, X_n^2 - X_n \rangle$ with an ideal $\langle X_1^2 - X_1, \ldots, X_n^2 - X_n \rangle$ also becomes a boolean ring, which is called a *boolean polynomial ring*. A Gröbner basis in a boolean polynomial ring, called a *boolean Gröbner basis*, is first introduced in [3, 4] with its computation algorithm in order to solve certain types of constraints over sets. The algorithm is implemented in [5, 6] for the case that $\mathbf{B}$ is a boolean ring that consists of all finite or co-finite subsets of S, here S is a set of all strings of the computer language. The program is released as a free software of ICOT(Institute for New Generation Computer Technology), however, it has been used by very few people. It is written in a uncommon computer language such as Prolog or Klic(a parallel logic programming language developed in ICOT).

After the release of [6], further theoretical developments are done in [1, 7, 9]. Based on these results, we implemented a new algorithm to compute boolean Gröbner bases in the computer algebra system Risa/Asir [2]. Our program achieves tremendous speed-up comparing with the old implementation of [6]. It enables us to do our recent work [8] of a non-trivial application of boolean Gröbner bases. The following table contains computation time(in terms of seconds) of 7 boolean Gröbner bases for solving 7 Sudoku puzzles which are ranked as extremely difficult. The row Risa/Asir and Klic cotain computation times of the same boolean Gröbner basis in each column by our new program and by the old program of [6] respectively, the symbol ∞ means that the computation did not terminate within 2 hours. All computations are done by a PC with 2GB memory and Core2Duo2GHZ CPU.

puzzle	1	2	3	4	5	6	7
Risa/Asir	41.7	43.6	48.1	40.1	44.3	48.9	76.2
Klic	134.1	398.3	1025.3	∞	1242.3	686.5	∞

In the poster, we introduce our new implementation together with our non-trivial Sudoku solver based on boolean Gröbner bases computation.

References

[1] Inoue, S. (2009). On the Computation of Comprehensive Boolean Gröbner Bases. Submitted for publication.

[2] Noro, M. et al. (2009). A Computer Algebra System Risa/Asir.
http://www.math.kobe-u.ac.jp/Asir/asir.html

[3] Sakai, K. and Sato, Y. (1988). Boolean Gröbner bases. ICOT Technical Momorandum 488.
http://www.icot.or.jp/ARCHIVE/Museum/TRTM/tm-list-E.html

[4] Sakai, K., Sato, Y. and Menju, S. (1991). Boolean Gröbner bases(revised). ICOT Technical Report 613.
http://www.icot.or.jp/ARCHIVE/Museum/TRTM/tr-list-E.html

[5] Sato, Y. et al.(1996). Set Constrains Solvers(Prolog version).
`http://www.icot.or.jp/ARCHIVE/Museum/FUNDING/funding-96-E.html`

[6] Sato, Y. et al.(1998). Set Constrains Solvers(Klic version).
`http://www.icot.or.jp/ARCHIVE/Museum/FUNDING/funding-98-E.html`

[7] Sato, Y. and Inoue, S.(2005). On the Construction of Comprehensive Boolean Gröbner Bases. Proceedings of the Seventh Asian Symposium on Computer Mathematics(ASCM2005), pp 145-148.

[8] Sato, Y., Inoue, S., Suzuki, A. and Nabeshima, K. Boolean Gröbner Bases and Sudoku. Submitted for publication.

[9] Sato, Y., Nagai, A. and Inoue, I.(2008). On the Computation of Elimination Ideals of Boolean Polynomial Rings, LNAI 5081, pp 334-348, Springer-Verlag Berlin Heidelberg.

Extending Cardinal's algorithm to a broader class of structured matrices

Claude-Pierre Jeannerod[a] Christophe Mouilleron[a] Gilles Villard[a]

[a] CNRS - INRIA - Université de Lyon (LIP / ENS Lyon)
e-mails: `claude-pierre.jeannerod@ens-lyon.fr` `gilles.villard@ens-lyon.fr`
`christophe.mouilleron@ens-lyon.org`

The cost of linear algebra (linear system solving, inversion,...) over general dense matrices of dimension n is $O(n^\omega)$ field operations, with $2 \leq \omega < 2.38$. However, the underlying problems sometimes have a structure such that the associated matrices, while still being dense, can be stored using fewer than n^2 field coefficients. Thus, for such problems, it is natural to search for algorithms subquadratic in n.

We will consider here one class of dense structured matrices. For a field $\mathbb{K}$, if $M \in \mathbb{K}^{n\times n}$ and $N \in \mathbb{K}^{n\times n}$, we will say that a matrix $A \in \mathbb{K}^{n\times n}$ is structured with respect to the operator $\nabla_{M,N} : A \mapsto MA - AN$ if the rank of $\nabla_{M,N}(A)$ is small compared to n. This rank, noted α hereafter, is in fact a measure of the structure: the smaller it is, the more structured the matrix is. Having $\alpha = \operatorname{rank} \nabla_{M,N}(A)$ ensures that there exist two matrices $G, H \in \mathbb{K}^{n\times\alpha}$ such that $\nabla_{M,N}(A) = GH^T$. We will say that (G, H) is a pair of generators of A with respect to the operator $\nabla_{M,N}(A)$.

Storing G and H requires only $2\alpha n$ coefficients, which is small compared to the n^2 coefficients needed for A when α is small. One important property is that the structure is preserved by inversion (see for example [6]):

Theorem 1. *If A is invertible and structured for the operator $\nabla_{M,N}$ then A^{-1} is structured for the operator $\nabla_{N,M}$. More precisely, we have*

$$\operatorname{rank}\big(\nabla_{N,M}(A^{-1})\big) = \operatorname{rank}\big(\nabla_{M,N}(A)\big).$$

Moreover, if $\nabla_{M,N}(A) = GH^T$ then $\nabla_{N,M}(A^{-1}) = YZ^T$, with $Y = -A^{-1}G$ and $Z = A^{-T}H$.

For such structured matrices, well-known techniques are available for computing a compressed version of A^{-1} or linear system solutions using $O^{\tilde{}}(\alpha^2 n)$ field operations [6]. (Here and hereafter the $O^{\tilde{}}$ notation hides all logarithmic factors.) Such techniques are essentially variations of the divide and conquer approach of Morf [5] and Bitmead and Anderson [1]. In practice, they apply to important cases like Cauchy-like matrices where both M and N are diagonal and Vandermonde-like matrices where M is diagonal and N equals the transpose of the classical lower shift matrix $\mathbb{Z}_n$, whose entry (i, j) is 1 if $i = j + 1$, and 0 otherwise.

However, all these techniques recursively use a "compression" stage which, given matrices $G_1, H_1 \in \mathbb{K}^{n\times\beta}$ such that $G_1 H_1^T$ has rank $\alpha \leq \beta$, computes matrices G_2, H_2 having exactly α columns. One exception is an algorithm by Cardinal [3], which shows in the case of Cauchy-like matrices how to get rid of this compression step by judiciously exploiting the explicit form of the generators Y, Z in Theorem 1.

Our goal here is twofold: first, to bring attention to Cardinal's trick, which seems to have been very rarely cited so far; second, to show how to extend it to a broader class of structured matrices that includes Vandermonde-like matrices. In particular, we show the following result:

Theorem 2. *Let M be lower-triangular and N be upper-triangular, and let A be structured for the operator $\nabla_{M,N}$. Then one can compute generators for A^{-1} without any compression stage. Furthermore, in the case of Cauchy-like and Vandermonde-like matrices, this can be done in $O^{\sim}(\alpha^2 n)$ field operations.*

Although the compression stage can be performed routinely using fast Gaussian elimination (either using randomization [4] or deterministically [6, 2]), suppressing it results in algorithms that are simpler to write and simpler to analyze. For now, this is the main interest of the above result. Another potential advantage of this compression-free approach (which we have not investigated yet) is some practical speed-ups when solving Cauchy-like and Vandermonde-like linear systems.

References

[1] R. R. Bitmead and B. D. O. Anderson. Asymptotically fast solution of Toeplitz and related systems of linear equations. *LAA*, 34:103–116, 1980.

[2] A. Bostan, C.-P. Jeannerod, and É. Schost. Solving structured linear systems with large displacement rank. *Theoretical Computer Science*, 407(1:3):155–181, 2008.

[3] J. P. Cardinal. On a property of Cauchy-like matrices. *Comptes Rendus de l'Acadmie des Sciences - Series I - Mathematics*, 328(11):1089–1093, 1999.

[4] E. Kaltofen. Asymptotically fast solution of Toeplitz-like singular linear systems. In *ISSAC'94*, pages 297–304. ACM, 1994.

[5] M. Morf. Doubling algorithms for Toeplitz and related equations. *IEEE Conference on Acoustics, Speech, and Signal Processing*, pages 954–959, 1980.

[6] V. Y. Pan. *Structured Matrices and Polynomials*. Birkhäuser Boston Inc., 2001.

Balanced Dense Polynomial Multiplication on Multi-cores

Marc Moreno Maza[a] **Yuzhen Xie**[b]

[a] Ontario Research Centre for Computer Algebra, University of Western Ontario, London, Canada
e-mail: moreno@csd.uwo.ca

[b] Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA
e-mail: yxie@csail.mit.edu

1 Introduction

In symbolic computation, polynomial multiplication is a fundamental operation akin to matrix multiplication in numerical computation. We present efficient implementation strategies for FFT-based dense polynomial multiplication targeting multi-cores. We focus on polynomials over finite fields since the so-called *modular techniques* reduce all computations to such fields of coefficients. Due to the constraints of 1-D FFT over finite fields, we assume 1-D FFTs are computed in a *black box* program, which is not necessarily a parallel one. This allows us to employ non-standard FFT techniques such as Truncated Fourier Transform (TFT) [5] which for many cases has better time and memory efficiency.

We explore the parallelism in the row-column algorithm for multidimensional FFT computations. We show that *balanced input data* can maximize parallel speedup and minimize cache complexity for bivariate multiplication. However, unbalanced input data, which are common in symbolic computation, are challenging. We provide efficient

techniques, what we call *extension* and *contraction*, to reduce multivariate (and univariate) multiplication to *balanced bivariate multiplication*.

The techniques proposed in this paper are implemented in the Cilk++ language [2], which extends C++ to the realm of multi-core programming based on the multi-threaded model realized in [4]. The Cilk++ language is also equipped with a provably efficient parallel scheduler by work-stealing [1]. We use the serial C routines for 1-D FFT and 1-D TFT from the `modpn` library [6]. Our integer arithmetic modulo a prime number relies also on the efficient functions from `modpn`, in particular the improved Montgomery trick [8], presented in [7]. Our benchmarks demonstrate good speedup on multi-cores.

2 FFT-based Multivariate Multiplication

Let $\mathbb{K}$ be a field and f, $g \in \mathbb{K}[x_1 < \cdots < x_n]$ be polynomials. Define $d_i = \deg(f, x_i)$ and $d'_i = \deg(g, x_i)$, for all i. Assume there exists a primitive s_i-th root of unity $\omega_i \in \mathbb{K}$, for all i, where s_i is a power of 2 satisfying $s_i \geq d_i + d'_i + 1$. Then fg can be computed as follows.

***Step* 1.** Evaluate f and g at each point of the n-dimensional grid $((\omega_1^{e_1}, \ldots, \omega_n^{e_n}), 0 \leq e_1 < s_1, \ldots, 0 \leq e_n < s_n)$ via n-D FFT.

***Step* 2.** Evaluate fg at each point P of the grid, simply by computing $f(P)g(P)$,

***Step* 3.** Interpolate fg (from its values on the grid) via n-D FFT.

3 Complexity Estimates

We consider the parallel running time of the above algorithm with the multi-threaded programming model of [4]. Our cache complexity estimates is based on the theoretical model introduced in [3].

Let $s = s_1 \cdots s_n$. The number of operations in $\mathbb{K}$ for computing fg based on FFTs is

$$\frac{9}{2}\sum_{i=1}^{n}(\prod_{j\neq i} s_j)s_i \lg(s_i) + (n+1)s = \frac{9}{2}s\lg(s) + (n+1)s.$$

Under our serial 1-D FFT assumption, the span of *Step* 1 is $\frac{9}{2}(s_1\lg(s_1) + \cdots + s_n\lg(s_n))$, and the parallelism of *Step* 1 is lower bounded by

$$s/\max(s_1, \ldots, s_n). \tag{4}$$

Let L be the size of a cache line. For some constant $c > 0$, the number of cache misses of *Step* 1 is upper bounded by

$$n\frac{cs}{L} + cs(\frac{1}{s_1} + \cdots + \frac{1}{s_n}). \tag{5}$$

Remark. For $n \geq 2$, Expression (5) is minimized at $n = 2$ and $s_1 = s_2 = \sqrt{s}$. Moreover, when $n = 2$, under a fixed $s = s_1 s_2$, Expression (4) is maximized at $s_1 = s_2 = \sqrt{s}$.

4 Contraction to Bivariate from Multivariate and Extension of Univariate to Bivariate

We describe below the techniques of contraction and extension by examples.

Given $f \in \mathbb{K}[x, y, z]$ where $\mathbb{K} = \mathbb{Z}/41\mathbb{Z}$, with $\deg(f, x) = \deg(f, y) = 1$, $\deg(f, z) = 3$. Contracting $f(x, y, z)$ to $f'(u, v)$ can be done by $x^{e_1}y^{e_2} \mapsto u^{e_1+2e_2}$ and $z^{e_3} \mapsto v^{e_3}$.

Consider $f, g \in \mathbb{K}[x]$ univariate, with $\deg(f) = 7$ and $\deg(g) = 8$; fg has "dense size" 16. We obtain an integer b, such that fg can be performed via $f_b g_b$ using "nearly square" 2-D FFTs, where $f_b := \Phi_b(f)$, $g_b := \Phi_b(g)$ and

$$\Phi_b : \ x^e \longmapsto u^{e \text{ rem } b}\ v^{e \text{ quo } b}.$$

Here $b = 3$ works since $\deg(f_b g_b, u) = \deg(f_b g_b, v) = 4$; moreover, the dense size of $f_b g_b$ is 25.

Proposition. For any non-constant $f, g \in \mathbb{K}[x]$, one can always compute b such that $|deg(f_b g_b, u) - deg(f_b g_b, v)| \leq 2$ and the dense size of $f_b g_b$ is at most twice that of fg.

5 Balanced Multiplication

We define that a pair of bivariate polynomials $p, q \in \mathbb{K}[u, v]$ is *balanced* if $\deg(p, u) + \deg(q, u) = \deg(p, v) + \deg(q, v)$.

Algorithm. Let $f, g \in \mathbb{K}[x_1 < \ldots < x_n]$. W.l.o.g. one can assume $d_1 >> d_i$ and $d_1{}' >> d_i$ for $2 \leq i \leq n$ (up to variable re-ordering and contraction). We obtain fg by

Step 1. Extending x_1 to $\{u, v\}$.

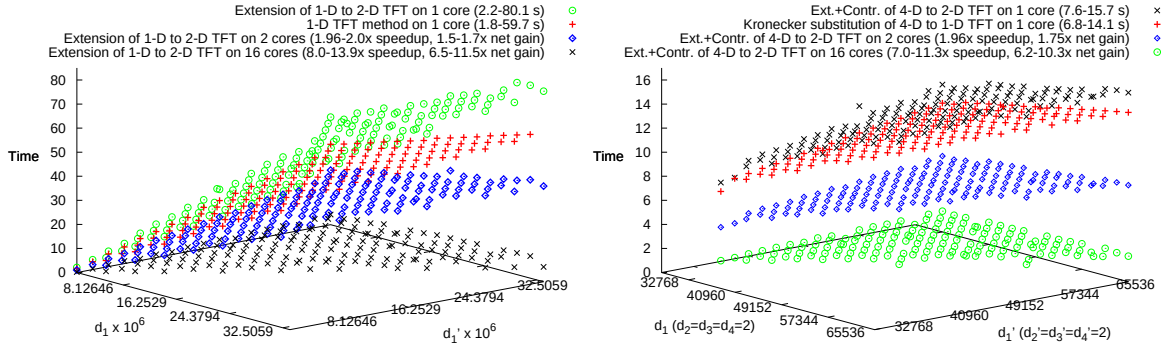
Step 2. Contracting $\{v, x_2, \ldots, x_n\}$ to v.

Determine the above extension Φ_b such that f_b, g_b is (nearly) a balanced pair and $f_b g_b$ has dense size at most twice that of fg.

6 Experimental Results

All our benchmarks are carried out on a 16-core machine with 16 GB memory and 4096 KB L2 cache. The processors are Intel Xeon E7340 @ 2.40GHz.

Aiming at supporting higher-level parallel algorithms for solving systems of non-linear equations, our multiplication must perform efficiently in terms of serial running time, parallelism and cache complexity, on any possible input degree patterns, insisting on those which put code efficiency to challenge. Our experimental results demonstrate that our balancing techniques are not only theoretical but also practical, as shown in the following figures.



References

[1] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. In *IEEE FOCS94*, 1994.

[2] Cilk Arts. Cilk++. `http://www.cilk.com/`.

[3] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science*, pages 285–297, 1999.

[4] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the cilk-5 multithreaded language. In *ACM SIGPLAN*, 1998.

[5] J. van der Hoeven. Truncated Fourier transform. In *Proc. ISSAC'04*. ACM Press, 2004.

[6] X. Li, M. Moreno Maza, R. Rasheed, and É. Schost. The modpn library: Bringing fast polynomial arithmetic into maple. In *MICA'08*, 2008.

[7] X. Li, M. Moreno Maza, and É Schost. Fast arithmetic for triangular sets: From theory to practice. In *Proc. ISSAC'07*, pages 269–276, NY, USA, 2007. ACM Press.

[8] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

A continued fraction type method to find a rational number in a given closed interval whose denominator is minimal

Hiroshi MURAKAMI[a]

[a] Department of Mathematics and Information Sciences, Tokyo Metropolitan University, 1-1, Minami-Osawa, Hachi-Oji, Tokyo, 192-0397, Japan
e-mail: mrkmhrsh@tmu.ac.jp

Abstract

We show a fast algorithm to find a rational number in a given real interval whose denominator is minimal. The algorithm is similar to the regular continued fraction expansion for a real number.

Keywords: rational number, interval, approximation, continued fraction

1 Definition of the Problem

Problem: For a given positive real closed interval $I = [\alpha, \beta]$, where $0 < \alpha < \beta$, find the rational number $r = p/q \in I$ whose denominator $q(> 0)$ is minimal.

We have restricted the interval to be positive without loss of generality. Because in cases α, β are in opposite signs or one of them is zero, the solution is trivial $r = 0, (p, q) = (0, 1)$. In case both are negatives, the solution is easily reduced to solve the case of positive interval $I = [-\beta, -\alpha]$ and then sign of the numerator of the solution is changed.

2 The trivial simplest solution

Let our inputs are $\alpha, \beta \in \mathbf{R}$, where $0<\alpha<\beta$. The problem requires to find the pair of positive integers (p, q) whose denominator q is minimal such that $\alpha \le p/q \le \beta$ is satisfied.

Since any closed interval whose length is not zero always contains rational numbers, and the rational number is countable, the value of denominator is bounded from below, therefore solution always exist and can be made constructive. For example, the following simple trivial procedure find one. The halt property of the procedure and the minimality of the denominator q are trivial from the construction. When the procedure below halts, the required solution is given by the pair of p and q as $r = p/q$.

```
L0 : for q ← 1, 2, 3, ... do
   if (interval I = [qα, qβ] contains an integer)then
      p ← any integer which is contained in I; exit L0
   endif
enddo L0
```

We can also understand easily that if we replaced "any integer" to "all integers" in the above procedure, all rational numbers whose denominators are the minimal value can be obtained. The statement "the interval $[q\alpha, q\beta]$ contains some integer" is tested by ceil($q\alpha$)$\le$floor($q\beta$).

This procedure relies on only the correctness of the determinations of whether $q\alpha$ and $q\beta$ are integer or of taking their integer part by rounding up or down, therefore it is not so difficult to realize on the computer (especially when both α, β are given by fraction expansion of finite length or they are given by rationals. For the case of algebraic numbers is also possible).

However, since this simple trivial method examines candidates of the denominator one by one incrementally, it is too much time consuming when the denominator of the solution is a huge number. Therefore, in the following more efficient algorithm will be researched.

3 An efficient method

For the positive interval $I = [\alpha, \beta]$, we may change from the statement of the original problem as "find some rational numbers contained in I whose denominators are minimal" to the following statement as "find the rational number whose numerator is minimal among those rational numbers contained in I whose denominators are minimal". (Because, if we only knew the minimal value of q, the numerator p will be any integer contained in $[q\alpha, q\beta]$ from the inequality of the interval; p is in the range $\text{ceil}(q\alpha), \ldots, \text{floor}(q\beta)$.)

When we change the problem as above, the following lemma 1 can be used.

Lemma 1. For the positive closed interval $I=[\alpha, \beta]$ where $0<\alpha<\beta$, the rational numbers (both numerator and denominator are non-negative integers) by the following two definitions always coincide. *Def1:* The rational number contained in I, and the denominator is minimal (if there are many such numbers, choose the one whose numerator is minimal). *Def2:* The rational number contained in I, and the numerator is minimal (if there are many such numbers, choose the one whose denominator is minimal).

In the paper, we call this unique rational number P/Q corresponding to the positive closed interval I as "the rational number which represents I".

Corollary 1. If the positive closed interval I contains any integer, let n be the minimal such integer, then the integer n is the rational number which represents I.

From the above lemma 1 and corollary 1, we get the principle of the continued fraction expansion type algorithm to find the rational number which represents I.

Principle of Continued Fraction Expansion type algorithm Let r be the rational number represents the positive closed interval $I=[\alpha, \beta]$. We first let $\alpha_0 \equiv \alpha$, $\beta_0 \equiv \beta$, $r_0 \equiv r$.

In the k-th step, we let the positive closed interval as $I_k \equiv [\alpha_k, \beta_k]$, and let r_k be the rational number which represents I_k.

When I_k contains some integer, let the minimal value of such integer be n_k, then (by the corollary 1) the integer n_k is the rational number r_k which represents I_k.

When I_k contains no integer, then α_k is not a integer and let $m_k \leftarrow \text{floor}(\alpha_k)$ then $0<\alpha_k-m_k<1$, therefore we have $0<\alpha_k-m_k \le r_k-m_k \le \beta_k-m_k$. From this inequality, we have the relation of the reciprocals: $0<1/(\beta_k-m_k) \le 1/(r_k-m_k) \le 1/(\alpha_k-m_k)$. If we let $\alpha_{k+1} \equiv 1/(\beta_k-m_k)$, $\beta_{k+1} \equiv 1/(\alpha_k-m_k)$, $r_{k+1} \equiv 1/(r_k-m_k)$, then $0<\alpha_{k+1}<\beta_{k+1}$, and we let the interval $I_{k+1} \equiv [\alpha_{k+1}, \beta_{k+1}]$ then we know that I_{k+1} contains the rational number r_{k+1}. In that situation, the following lemma 2 holds.

Lemma 2. The above transformation maps the positive closed interval I_k to another positive closed interval I_{k+1}, and a rational number r_k in the interval I_k to another rational number r_{k+1} in the interval I_{k+1}. When r_k is the rational number which represents I_k, then r_{k+1} is also the rational number which represents I_{k+1}.

From the above lemma 2, the both ends of the original and mapped intervals and also the representatives of the both intervals are related by a linear fractional transformation (Möbius transformation). If we know the integer m_k and the rational number r_{k+1} which represents I_{k+1}, then the rational number r_k which represents I_k can be calculated by the backward transformation as $r_k = m_k + 1/r_{k+1}$.

We continue the forward transformations until at the $k = \ell$-th step the interval I_ℓ contains an integer n_ℓ. Then the integer n_ℓ is the rational number which represents I_ℓ and we terminate the transformation. And we start from $r_\ell = n_\ell$ and repeat the backward transformations to obtain the desired solution r_0.

If we have $(m_0, m_1, \cdots, m_{\ell-1})$ and n_ℓ as the expansion, then the solution is calculated by the regular continued fraction as $r_0 = [m_0, m_1, m_2, \cdots, m_{\ell-1}, n_\ell] = m_0 + 1/(m_1 + 1/(m_2 + \cdots + 1/(m_{\ell-1} + 1/n_\ell)\cdots))$.

Therefore, the solution r_0 which we want to solve can be given by the following algorithm:

```
r_ℓ ← n_ℓ ; for j ← ℓ − 1 downto 0 do
    r_j ← m_j + 1/r_{j+1}
enddo
```

or if we represent the numerators as P_j and denominators as Q_j,$r_0=P_0/Q_0$ can be calculated by the following algorithm:

```
P_ℓ ← n_ℓ ; Q_ℓ ← 1;
for j ← ℓ − 1 downto 0 do
    P_j ← m_j P_{j+1} + Q_{j+1} ; Q_j ← P_{j+1}
enddo
```

Lemma 3. Coefficients of the expansion are not less than 1 except the first coefficient.
When the expansion terminates at the ℓ-th step, the integer n_ℓ contained in the interval I_ℓ is not less than 1. And the expansion continued both k-th and $(k+1)$-th steps, then $m_{k+1}\geq 1$. Therefore, the coefficients m_j in the continued fraction expansion is no less than 1 except the first m_0.

In the conclusion, if the expansion terminates at the ℓ-th step, the rational number which represents I is the regular continued fraction $[m_0, m_1, \cdots, m_{\ell-1}, n_\ell]$ whose coefficients except m_0 are no less than 1.

Lemma 4. The expansion terminates in finite steps. When the transformation is continued, eventually the interval which contains integer is reached in the finite steps and the expansion terminates.

Continued Fraction Expansion type algorithm: For the given positive real numbers $0<\alpha<\beta$, the algorithm below solves the rational r which represents the positive closed interval $[\alpha, \beta]$.

```
α_0 ← α; β_0 ← β;
L0:for k ← 0, 1, 2, ··· do
  if (I_k = [α_k, β_k] contains some integer) then
    r_k ← the minimal integer in I_k exit L0
  else
    m_k ← integer part of α_k;
    α_{k+1} ← 1/(β_k − m_k) ; β_{k+1} ← 1/(α_k − m_k)
  endif
enddo L0 ;
while (k ≥ 0) do
  r_{k−1} ← m_{k−1} + 1/r_k ; k ← k − 1
enddo ;
r ← r_0
```

The solution whose denominator is a large number can be obtained much faster by this present method a variant of the regular continued fraction expansion than the trivial one.

For the continued fraction expansion finished at the ℓ-th step, the possible smallest value of the denominator is $F_{\ell+1}$ which corresponds to the case of expansion $[m_0, 1, 1, \cdots, 1]$ (there are ℓ ones). Where, F_n is the Fibonacci number defined by F_1=1,F_2=1,$F_{n+2}=F_n+F_{n+1}$ and $F_n \approx \phi^n/\sqrt{5}$,$\phi=(1+\sqrt{5})/2\approx 1.618$.

Let q be the denominator of the true solution. The trivial method searches the denominator incrementally from 1 to q. Therefore the number of arithmetics $O(q)$ is exponential to the bit length of q. On the other hand, the continued fraction expansion method terminates at the step ℓ which is k or less, where k is the minimal number which satisfies $F_{k+1}\geq q$. Since $k+1\approx \log(\sqrt{5}q)/\log\phi$, the number of arithmetics of the present method is $O(\log q)$ linear to the bit length of q.

Example: The PC system used is CPU intel Core i7 920 (2.67GHz) using 1 thread with intel Fortran v11.0 optimization flag "-fast" with IEEE 64bit floating number. For the interval $[\alpha, \beta]$=[0.217973763834575, 0.217973763834580], the trivial method in 60 millisecond gave the answer p=3493241, q=16025970, p/q=0.217973763834576. The present method made the expansion [0,4,1,1,2,2,1,5,1,6,66,1,1,1,1,2,1,3], and gave the same rational p=3493241, q=16025970 in total 0.40 microsecond (which is too short for the single measurement, so this time is the average value of the repeat of 1000 calls).

Determining Divisibility between Polynomials with Inexact Coefficients

Hiroki Nakayama[a1] Hiroshi Sekigawa[a]

[a] NTT Communication Science Laboratories, Nippon Telegraph and Telephone Corporation
e-mail: h-nakayama@cs.brl.ntt.co.jp
e-mail: sekigawa@theory.brl.ntt.co.jp

Abstract

We provide a method of determining whether there exist some $p \in P$ and $f \in F$ such that p is divisible by f for a pair of real multivariate interval polynomials, P and F. Although this problem is written as a feasibility problem for a system of nonlinear algebraic equations, it is an NP-hard problem and is thus difficult to solve. Our approach is iterative based on interval analyses, which outputs the regions containing the solutions if the system is feasible; otherwise, it outputs the fact of infeasibility. We also propose two methods for where the system of algebraic equations is underdetermined, the first obtains the regions that enclose all solutions, and the second obtains the solution that minimizes error in the Euclidean norm.

1 Introduction

We consider the following type of problem and provide an iterative method to solve it.

Problem 1. For given $l_\alpha, h_\alpha, l'_\beta, h'_\beta \in \mathbb{R}$, do there exist real multivariate polynomials

$$p = \sum_\alpha a_\alpha x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_k^{\alpha_k} \quad (\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_k)),$$
$$f = \sum_\beta b_\beta x_1^{\beta_1} x_2^{\beta_2} \cdots x_k^{\beta_k} \quad (\beta = (\beta_1, \beta_2, \ldots, \beta_k)),$$

where $l_\alpha \le a_\alpha \le h_\alpha$, $l'_\beta \le b_\beta \le h'_\beta$, and $f \mid p$?

Problem 1 can be treated as a feasibility problem with polynomial equations through similar arguments in [8]. When the divisor f is an ordinary polynomial, i.e., every b_β is fixed, all constraints on polynomial equations are linear, thus the problem is easily solved using the properties of zonotopes [8]. But in general case, Problem 1 is NP-hard and thus difficult (for the complexity of the algorithm, see Chapter 13 of [2]). In our iterative method based on interval analyses, each step is done by applying interval Gaussian elimination, thus it is expected to give the answer for the problem efficiently.

2 Description of Problems

For the details of interval analyses, see [4]. We denote the domain of the interval coefficients by $\mathbb{IR}$ and that of the interval polynomials by $\mathbb{IR}[\mathbf{x}]$ $(\mathbf{x} = (x_1, \ldots, x_k))$. For given interval polynomials $P = \sum_{i=1}^{s} a_i e_i(\mathbf{x})$ and $F = \sum_{j=1}^{t} b_j d_j(\mathbf{x})$ with $a_i, b_j \in \mathbb{IR}$ and $e_i(\mathbf{x}), d_j(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]$, Problem 1 is rewritten as follows.

Problem 2. Determine whether there exist some polynomials $p \in P$ and $f \in F$ such that $f \mid p$.

We regard interval coefficients $a_1, \ldots, a_s$ and $b_1, \ldots, b_t$ as symbols, then consider the remainder, $\mathrm{rem}(P, F)$. Expanding and sorting P and F in arbitrary term order $\succ$, we write $P = \sum_\alpha f_\alpha(a_1, \ldots, a_s) \cdot \mathbf{x}^\alpha$ and $F = \sum_\alpha g_\alpha(b_1, \ldots, b_t) \cdot \mathbf{x}^\alpha$. Then, the remainder polynomial, $P' = \mathrm{rem}(P, F)$, is described as

$$P' = \sum_\alpha c_\alpha(a_1, \ldots, a_s, b_1, \ldots, b_t) \cdot \mathbf{x}^\alpha.$$

The condition where $P' = 0$ is regarded as all coefficients $c_\alpha(a_1, \ldots, a_s, b_1, \ldots, b_t)$ having vanished for some $a_1, \ldots, a_s$ and $b_1, \ldots, b_t$ contained in the intervals. Therefore, Problem 2 is formulated as a feasibility problem of an algebraic equation system as

Problem 3. *For the algebraic equation system*

$$\{c_\alpha(a_1, \ldots, a_s, b_1, \ldots, b_t) = 0 \mid \text{for each term of } P'\},$$

does any solution exist satisfying $l_i \le a_i \le h_i$ and $l'_j \le b_j \le h'_j$ $(i = 1, \ldots, s;\ j = 1, \ldots, t)$?

[1] The current affiliation of the author is: NEC Corporation, h-nakayama@cj.jp.nec.com

3 Iterative Algorithms

3.1 Interval Newton Methods

In the following, we treat the feasibility problem of an algebraic equation system (Prob. 3) instead of the divisibility problem between interval polynomials (Prob. 2). Let the variables of a system $\mathbf{c} = (c_1, \ldots, c_n)$ and its constraints $\mathbf{f} = (f_1, \ldots, f_m)$, then the following problem is obtained.

$$\begin{aligned} \text{find} \quad & \mathbf{c} = (c_1, \ldots, c_n) \\ \text{s.t.} \quad & f_j(\mathbf{c}) = 0 \quad (j = 1, \ldots, m), \\ & l_i \le c_i \le h_i \quad (i = 1, \ldots, n). \end{aligned} \tag{6}$$

We have denoted the number of constraints by m and that of variables by n. In this section, we deal with problems in the case of $m \ge n$, where the number of solutions is finite. Conversely, the case of $m < n$ is described in Section 4.

We adopt the Moore's approach [7] based on the interval Newton method, i.e., narrowing the rectangular region that contains some accurate solutions and finally outputting a sufficiently reduced region as an interval, instead of the accurate solution itself.

For a function, $\mathbf{f} : \mathbb{IR}^n \to \mathbb{IR}^m$, and an interval, $\mathbf{c^I} \in \mathbb{IR}^n$, the step that improves the solution from $\mathbf{c^I}$ to $\mathbf{d^I}$ is

$$\mathbf{0} = \mathbf{f}(\mathbf{c}) + \mathbf{J}(\mathbf{c}, \mathbf{c^I}) \cdot (\mathbf{d^I} - \mathbf{c}), \tag{7}$$

where $\mathbf{c}$ means a point chosen from $\mathbf{c^I}$. We generally adopt the midpoint, $\mathrm{mid}(\mathbf{c^I})$. $\mathbf{J}(\mathbf{c}, \mathbf{c^I}) \in \mathbb{IR}^{m \times n}$ is a Jacobian matrix of $\mathbf{f}$, described as

$$J_{ij}(\mathbf{c}, \mathbf{c^I}) = \tfrac{\partial f_i}{\partial c_j}(c_1^{\mathbf{I}}, \ldots, c_j^{\mathbf{I}}, c_{j+1}, \ldots, c_n).$$

For each entry of $\mathbf{J}(\mathbf{c}, \mathbf{c^I})$, we substitute a value in the interval for the variable c_i and the interval itself for the variable $c_i^{\mathbf{I}}$, and evaluate the lower and upper bounds using affine arithmetic [1, 6]. By partially replacing the substitutions of intervals with points and using affine arithmetic, the spread of intervals in $\mathbf{J}(\mathbf{c}, \mathbf{c^I})$ are reduced.

In Eq. (7), the values of $\mathbf{c}$, $\mathbf{f}(\mathbf{c})$, and $\mathbf{J}(\mathbf{c}, \mathbf{c^I})$ are already known, thus the value of $\mathbf{d^I}$ is obtained by solving a linear interval equation. We obtain a rectangular region that properly contains an accurate region using interval Gaussian elimination [3] and regard the solution as relaxed.

Before applying interval Gaussian elimination, we normalize the interval matrix. For a matrix, $\mathbf{B} \in \mathbb{R}^{m \times m}$, such that $\mathrm{mid}(\mathbf{M}) = \mathbf{B} \cdot \mathrm{mid}(\mathbf{J})$ is an identity matrix (if $m > n$, $\mathrm{mid}(M_{ii}) = 1$, and $\mathrm{mid}(M_{ij}) = 0$ for $i \ne j$), we call multiplication of $\mathbf{B}$ to both sides of Eq. (7) normalization. Writing $\mathbf{M}(\mathbf{c}, \mathbf{c^I}) = \mathbf{B} \cdot \mathbf{J}(\mathbf{c}, \mathbf{c^I})$, $\mathbf{r}(\mathbf{c}) = -\mathbf{B} \cdot \mathbf{f}(\mathbf{c})$, and $\mathbf{N}(\mathbf{c}^{(k)}, \mathbf{c}^{\mathbf{I}(k)}) = \mathbf{d^I}$ to clarify that Eq. (8) is an iteration step to obtain $\mathbf{c}^{\mathbf{I}(k+1)}$ from $\mathbf{c}^{\mathbf{I}(k)}$ to find the solutions to Eq. (7), the step is described as

$$\mathbf{M}(\mathbf{c}^{(k)}, \mathbf{c}^{\mathbf{I}(k)}) \cdot \left(\mathbf{N}(\mathbf{c}^{(k)}, \mathbf{c}^{\mathbf{I}(k)}) - \mathbf{c}^{(k)} \right) = \mathbf{r}(\mathbf{c}^{(k)}), \tag{8}$$

$$\mathbf{c}^{\mathbf{I}(k+1)} = \mathbf{c}^{\mathbf{I}(k)} \cap \mathbf{N}(\mathbf{c}^{(k)}, \mathbf{c}^{\mathbf{I}(k)}). \tag{9}$$

$\mathbf{c}^{\mathbf{I}(k+1)} \subseteq \mathbf{c}^{\mathbf{I}(k)}$ is obviously satisfied by the definitions of Eqs. (8) and (9).

3.2 Termination of Iterations

We need to determine some termination conditions to end the algorithm within a finite number of steps. Following two conditions are tested at the end of each step:

1. There is no solution in the region, i.e., the current box becomes empty after the iterations.
2. There exist some solutions in the region. We then output the current region as a solution.

The criteria are given as the following statements:

Theorem 1 ([4]). *There is no solution in the discarded region,* $\mathbf{c}^{\mathbf{I}(k)} \setminus \mathbf{c}^{\mathbf{I}(k+1)}$. *Particularly if* $\mathbf{c}^{\mathbf{I}(k+1)} = \emptyset$, *there are no zeros of* $\mathbf{f}$ *in* $\mathbf{c}^{\mathbf{I}(k)}$.

Theorem 2 ([4]). *If* $\mathbf{N}(\mathbf{c}^{(k)}, \mathbf{c}^{\mathbf{I}(k)}) \subseteq \mathbf{c}^{\mathbf{I}(k)}$, *some zeros of* $\mathbf{f}$ *are in* $\mathbf{N}(\mathbf{c}^{(k)}, \mathbf{c}^{\mathbf{I}(k)})$.

By using these arguments, the algorithm to enumerate solutions in Eq. (6) for $m \geq n$ is given as follows.

Algorithm 1.
Input: Set of polynomials $\mathbf{f}(\mathbf{c}) = (f_1, \ldots, f_m)$ *and initial box* ${\mathbf{c}^{\mathbf{I}}}_{init}$ *of* $\mathbf{c} = (c_1, \ldots, c_n)$.
Output: List of intervals containing zeros of $\mathbf{f}$. *If no solution exists, return empty set.*
1. $L = \{{\mathbf{c}^{\mathbf{I}}}_{init}\}$.
2. *Pick an element from* L *and denote it by* $\mathbf{c}^{\mathbf{I}(k=0)}$.
3. *Evaluate* $\mathbf{f}(\mathbf{c}^{\mathbf{I}(k)})$ *using affine arithmetic. If the result does not contain* $\mathbf{0}$, *no solution is in* $\mathbf{c}^{\mathbf{I}(k)}$. *Go to 2.*
4. *Using normalization and interval Gaussian elimination, obtain* $\mathbf{c}^{\mathbf{I}(k+1)}$ *from* $\mathbf{c}^{\mathbf{I}(k)}$.
5. *If* $\mathbf{c}^{\mathbf{I}(k+1)} = \emptyset$, *no solution is in* $\mathbf{c}^{\mathbf{I}(k)}$. *Go to 2.*
6. *If* $\mathbf{c}^{\mathbf{I}(k+1)} = \mathbf{c}^{\mathbf{I}(k)}$, *divide* $\mathbf{c}^{\mathbf{I}(k)}$ *into two sub-regions and add them into* L, *then go to 2.*
7. *If* $\mathbf{N}(\mathbf{c}, \mathbf{c}^{\mathbf{I}}) \subseteq \mathbf{c}^{\mathbf{I}(k)}$, *output the interval,* $\mathbf{c}^{\mathbf{I}(k+1)}$, *and go to 2. Otherwise, increment* k *by* 1 *and go to 3.*

4 Underdetermined Case

4.1 Setting Basic Variables

For $m < n$ in Eq. (6), we divide the set of variables, $\mathrm{var}(\mathbf{c}) = \{c_1, \ldots, c_n\}$, into two; the first is a set of basic variables $\mathrm{var}(\mathbf{c}_b) = \{c_{i_1}, \ldots, c_{i_m}\} \subset \mathrm{var}(\mathbf{c})$ and the second is that of non-basic variables $\mathrm{var}(\mathbf{c}_{nb}) = \mathrm{var}(\mathbf{c}) \setminus \mathrm{var}(\mathbf{c}_b)$. For such fixed bases, we find regions such that the solution to $\mathbf{c}_b$ is uniquely determined for every fixed $\mathbf{c}_{nb} \in {\mathbf{c}^{\mathbf{I}}}_{nb}$. This approach is based on [5].

The algorithm consists of two phases. In the former, the region is divided and diminished through iterative steps with replacements of bases, and finally we obtain regions where by fixing any $\mathbf{c}_{nb} \in {\mathbf{c}^{\mathbf{I}}}_{nb}$, the system with m variables and m constraints only has one zero. In the latter, we obtain a set of regions enclosing all solutions whose radiuses are smaller than a threshold δ, with the basic variables remaining unchanged.

4.2 Optimization on Euclidean Norm

This section takes into consideration whether a solution can be found that minimizes its Euclidean distance from the center. This problem is regarded as an optimization problem, thus formulated as

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{i=1}^{n} \big(c_i - (h_i + l_i)/2\big)^2 \\ \text{s.t.} \quad & f_j(\mathbf{c}) = 0 \quad (j = 1, \ldots, m), \\ & l_i \leq c_i \leq h_i \quad (i = 1, \ldots, n). \end{aligned} \tag{10}$$

We use the John condition, which is a generalization of the KKT condition, to find the optimal solution (10). This condition describes the points that attain minimal values for given constraints $\mathbf{f}(\mathbf{c}) = \mathbf{0}$ and an objective function, $g(\mathbf{c})$. It is formulated as

$$\begin{aligned} u + \textstyle\sum_j E_j v_j &= 0, \\ u\dfrac{\partial g}{\partial c_i}(\mathbf{c}) + \textstyle\sum_j v_j \dfrac{\partial f_j}{\partial c_i}(\mathbf{c}) &= 0, \\ f_j(\mathbf{c}) &= 0, \end{aligned} \tag{11}$$

where $u \geq 0$, v_j are Lagrange multipliers and $E_j \in [1, 1+\epsilon]$ for all $j = 1, \ldots, m$. The constant, ϵ, is the smallest positive machine number such that in the number system used on computers. In Eq. (11), the top equality is a normalization condition, the middle one is a condition such that $g(\mathbf{c})$ is minimal, and the bottom one describes zeros of $\mathbf{f}$.

Since both the number of variables and constraints in the system of algebraic equations (11) are $m + n + 1$, the number of solutions is finite. Therefore, we first compute all solutions to (11) to find $\mathbf{c}$ that minimizes $g(\mathbf{c})$, then choose one where the value of g becomes minimum.

References

[1] M. V. A. Andrade, J. L. D. Comba and J. Stolfi, "Affine arithmetic," In *Proc. INTERVAL '94*, pp. 36–40, 1994.

[2] S. Basu, R. Pollack and M.-F. Roy, *Algorithms in Real Algebraic Geometry*, Springer-Verlag, 2nd Edition, 2006.

[3] A. Frommer. "A feasibility result for interval Gaussian elimination relying on graph structure," In *Symbolic Algebraic Methods and Verification Methods*, Springer, Wien, pp. 79–86, 2001.

[4] E. Hansen and G. W. Walster, *Global Optimization Using Interval Analysis: Revised and Expanded*, Marcel Dekker Inc., New York, 2nd Edition, 2003.

[5] Y. Kanzawa, M. Kashiwagi and S. Oishi, "An algorithm for iteratively refining the interval including the solution set of parameter dependent nonlinear equations," *IEICE Trans. Fundamentals (Jpn. Ed.)*, Vol. J83-A, No. 5, pp. 511–516, 2000.

[6] S. Miyajima, T. Miyata and M. Kashiwagi, "On the best multiplication of the affine arithmetic," *IEICE Trans. Fundamentals (Jpn. Ed.)*, Vol. J86-A, No. 3, pp. 232–240, 2003.

[7] R. E. Moore, "A test for existence of solutions to nonlinear systems," *SIAM J. Numer. Anal.*, Vol. 47, No. 4, pp. 611–615, 1977.

[8] H. Sekigawa, "On real factors of real interval polynomials," *J. Symbolic Comput.*, Vol. 44, No. 7, pp. 908–922, 2009.

A Vector-Quantization Approach to Coding Systems

Moshe Porat[a]

[a] Department of Electrical Engineering, Technion – Israel Institute of Technology, Technion City, Haifa 32000, ISRAEL
Phone: +972-48294684, Fax: +972-48324411, mp@ee.technion.ac.il

Abstract

The increasing amount of information transmitted over communication channels has to be handled efficiently to avoid congestion and delays in the network service. In this work, an algebraic approach to data coding and transmission is proposed. One of its basic assumptions is that in many situations the information could be decomposed into basic components or vectors, which represent a significant part of the data. To illustrate the capabilities of the proposed approach, a coding system based on vector quantization is investigated. This approach is based on the observation that in most systems, events of recent history of the stream can be used to represent future vectors of the data. Our conclusion is that the proposed algebraic approach could be efficient in presently used coding systems.

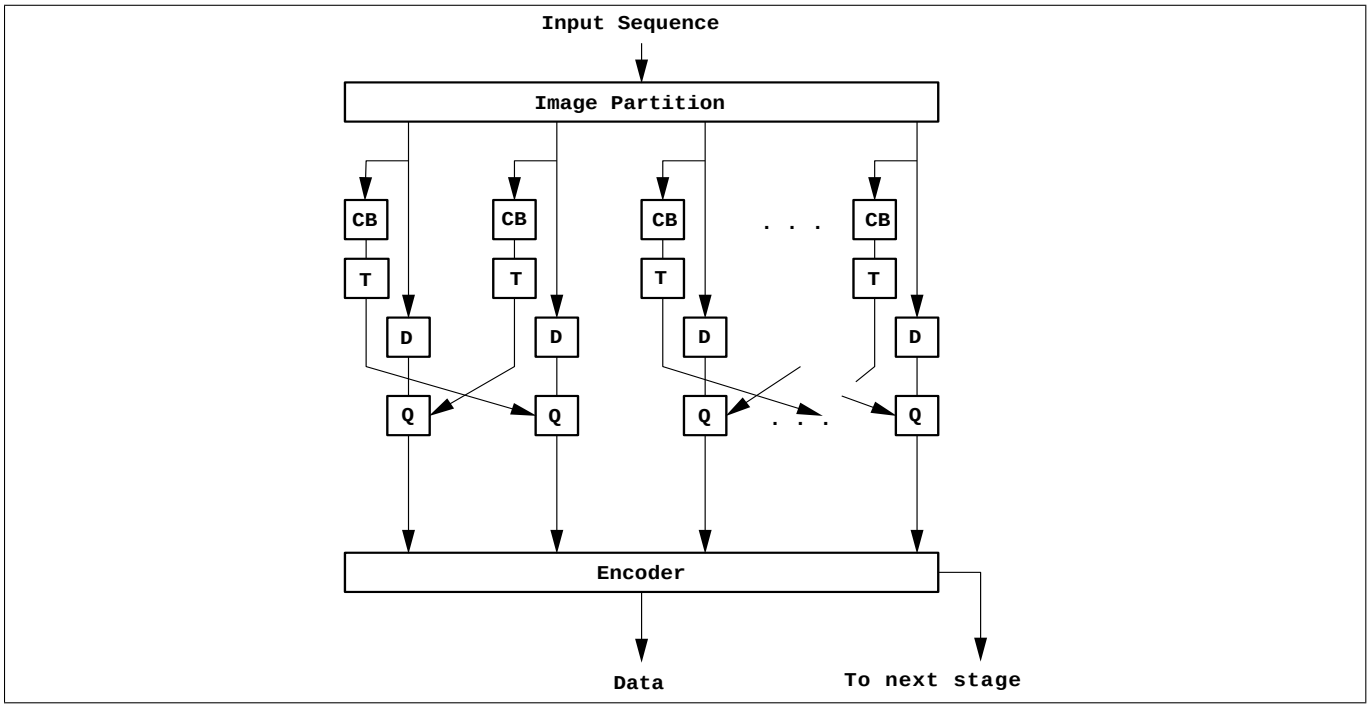


Figure 2: A schematic block diagram of the coding system.

5 Method

Wideband multimedia channels play an important role in today's information systems. Accordingly, a need exists for systems that could encode and decode efficiently and use information about the specific characteristics of the source and the user to reproduce the information with minimal distortion. The model used in this study is based on specific properties related to the nature of the data streams. In the case tested, video information is considered. In particular, the main assumption is that changes or events of recent history are likely to repeatedly appear in the same area of the image where they have previously appeared. If not similarly repeated (up to allowed distortion), they are likely to be encoded as *transformed* versions of previous data. According to this model, a frame is divided into blocks and sub-blocks.

Based on the above assumptions, the proposed system is presented in Figure 1. In the first stage of the process, the blocks of the image after Image Partition are used for training localized codebooks (CB) using vector quantization [1]. The Delay (D) is needed to allow updating of the codebooks before quantization (Q). Most of the sub-blocks (vectors) of each block are adequately encoded in the first stage by relatively sparse codebooks. Some of the vectors, however, require additional attention due to distortion above a pre-determined threshold. Usually most of these vectors can be encoded using adjacent codebooks. It is assumed that these vectors refer to motions that have crossed the borders of their block (codebook) thus can be found in one of the adjacent codebooks, possibly after Transformation (T) of rotation or scaling.

6 Results

The algebraic approach to data coding using the history of the information sequence provides practical applications. Experimental results indicate a high compression ratio of more than 100:1, obtaining almost lossless reproduction of the original data with less than 0.1% distortion. In addition to the quality of the transmission, the implementation of this approach can be systematically organized in a parallel manner by its very nature since different codebooks are created for each block and searched for independently. It should be noted that even for encoding by serial machines, reduced complexity is achieved by processing of many small codebooks instead of a combined one. Based on its performance, it is suggested that the new localized computational approach to data coding be further analyzed and integrated into presently available methods [2], [3].

References

[1] Linde, Y.L., Buzo, A., and Gray, R.M., An Algorithm for Vector Quantizer Design, IEEE-COM, 28, 84-95, 1980.

[2] Kirshner, H. and Porat, M., "On the Approximation of L2 Inner Products from Sampled Data", IEEE Trans. on Signal Processing, Vol. 55, No. 5, pp. 2136-2144, 2007.

[3] Nemirovsky, S. and Porat, M., "On Texture and Image Interpolation using Markov Models", Signal Processing: Image Communication, Volume 24, pp. 139-157, 2009.

A Slice Algorithm for Koszul Simplicial Complexes on the Lcm Lattice of Monomial Ideals

Bjarke Hammersholt Roune

The Koszul Complex and Lcm Lattice

Let I be a monomial ideal in a polynomial ring with indeterminates $x_1, \ldots, x_n$, and define the *lcm lattice*[5] by

$$\operatorname{lat}(I) := \{\operatorname{lcm}(M) \,|M \subseteq \min(I)\}.$$

The lattice $\operatorname{lat}(I)$ encodes the combinatorial structure of I, and interesting information about I such as resolutions, Hilbert series and irreducible decomposition are gainfully computed and understood in terms of this lattice. Often only local information around each point x^a on the lattice is necessary, and this is encoded by the *(upper) Koszul complex*, which is the abstract simplicial complex defined by

$$\Delta^I_{x^a} := \left\{v \in \{0,1\}^n \,\middle|\, x^{a-v} \in I\right\}, \text{ where } x^v \notin I \text{ for } v_i < 0.$$

In this abstract we present an algorithm that computes the Koszul complexes of points on the lcm lattice, and thus reduces questions about a general monomial ideal to questions about a single point on a square free monomial ideal. This leads to algorithms for Hilbert-Poincaré series (different from Bigatti et.al.'s algorithm [2, 1]), irreducible decomposition ([3], generalizing the Slice Algorithm [6]) and Koszul homology (as described by E. Saenz-de-Cabezon [4]).

We wish to thank Eduardo Saenz-de-Cabezon and Anna Maria Bigatti for helpful discussions on these topics.

The Staircase Subset

Those $m \in \operatorname{lat}(I)$ where $\max(\text{facets}(\Delta^I_m)) \neq 0$ are uninteresting for applications, so we ignore them. Thus the output of the algorithm is the set $\left\{(m, \Delta^I_m) \,|m \in \operatorname{turn}(I)\right\}$, where

$$\operatorname{turn}(I) := \left\{m \in \operatorname{lat}(I) \,\middle|\, \max(\text{facets}(\Delta^I_m)) = 0\right\}.$$

Slices and Content

The algorithm recursively considers smaller and smaller subsets of $\operatorname{turn}(I)$ in a divide-and-conquer fashion, until each subset is trivially simple. We represent each subset by a *slice*, which is a tuple (I, S, q) where I and S are monomial ideals and q is a monomial. A monomial m is then in the subset represented by (I, S, q) if $mx_1 \cdots x_n \in (\operatorname{turn}(I) \setminus S)q$. Thus each slice accounts for some part of the output, it's *content*, which is given by

$$\operatorname{con}(I, S, q) := \left\{(mq, \Delta^I_m) \,|mx_1 \cdots x_n \in \operatorname{turn}(I) \setminus S\right\}.$$

If I is the input ideal, then we initially consider the slice $(Ix_1 \cdots x_n, \langle 0\rangle, 1)$, since it accounts for all of the output, i.e. its content is $\left\{(m, \Delta^I_m) \,|m \in \operatorname{turn}(I)\right\}$.

Note how the multiplication by $x_1 \cdots x_n$ in the starting slice and in the definition of content cancel each other. This may seem to be an unnecessary complication, but it is essential since none of the equations given below hold without it.

Divide...

We split a slice into two simpler slices according to the equation

$$\operatorname{con}(I,S,q) = \operatorname{con}(I:p, S:p, qp) \ \cup \ \operatorname{con}(I, S+\langle p\rangle, q),$$

where p is some monomial such that $1 \neq p \notin S$ and the union is disjoint. I.e. we recursively replace a slice (I,S,q) by two simpler slices $(I:p, S:p, qp)$ and $(I, S+\langle p\rangle, q)$. We then apply the following equation to each of the two slices,

$$\operatorname{con}(I,S,q) = \operatorname{con}(I',S,q), \quad I' := \left\langle x^a \in \min(I) \,\middle|\, x^{a'} \notin S \right\rangle,$$

i.e. we discard elements x^a of $\min(I)$ if $x^{a'}$ lies in S, where $a'_i := \max(0, a_i - 1)$.

Note that these equations are the equations used in the original Slice Algorithm [6], and looking at that paper should make the ideas in this abstract more clear.

...and Conquer

The algorithm has two base cases. First, if x_i does not divide $\operatorname{lcm}(\min(I))$ for some x_i, then $\operatorname{con}(I,S,q) = \emptyset$. Second, if I is square free, then

$$\operatorname{con}(I,S,q) = \left\{ \left(q, \left\{ v \in \{0,1\}^n \,\middle|\, x^{(1,\ldots,1)-v} \in I \right\} \right) \right\}.$$

Pesudo-code

The pseudo-code below implements the algorithm.

function $\operatorname{con}(I,S,q)$

let $I' := \left\langle x^a \in \min(I) \,\middle|\, x^{a'} \notin S \right\rangle$

if $x_1 \cdots x_n$ does not divide $\operatorname{lcm}(\min(I'))$ then return $\emptyset$

if I' is square free then return $\left\{ \left(q, \left\{ v \in \{0,1\}^n \,\middle|\, x^{(1,\ldots,1)-v} \in I \right\} \right) \right\}$

let p be some monomial such that $1 \neq p \notin S$

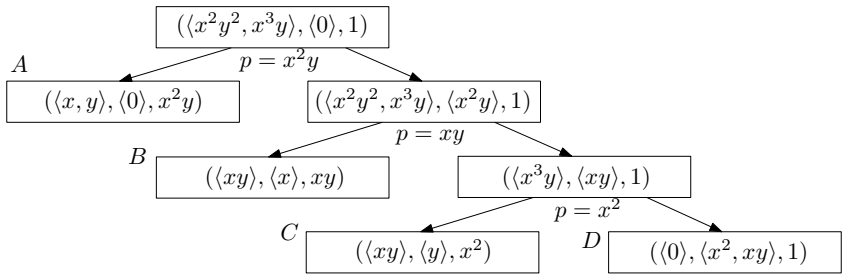
return $\operatorname{con}(I':p, S:p, qp) \ \cup \ \operatorname{con}(I', S+\langle p\rangle, q)$

Then the function call $\operatorname{con}(Ix_1 \cdots x_n, \langle 0\rangle, 1)$ returns $\left\{ (m, \Delta^I_m) \,|m \in \operatorname{turn}(I) \right\}$.

Example

The tree shows the computation on input $\langle xy, x^2\rangle$.

Computation tree



Output from leafs

$A : (x^2y, \{(0,0),(0,1),(1,0)\})$

$B : (xy, \{(0,0)\}$

$C : (x^2, \{(0,0)\}$

$D : \emptyset$

References

[1] Anna Maria Bigatti. Computation of Hilbert-Poincaré series. *J. Pure Appl. Algebra*, 119(3):237–253, 1997.

[2] Anna Maria Bigatti, Pasqualina Conti, Lorenzo Robbiano, and Carlo Traverso. A "divide and conquer" algorithm for Hilbert-Poincaré series, multiplicity and dimension of monomial ideals. In *Applied algebra, algebraic algorithms and error-correcting codes (San Juan, PR, 1993)*, volume 673 of *Lecture Notes in Comput. Sci.*, pages 76–88. Springer, Berlin, 1993.

[3] Anna Maria Bigatti and Eduardo Saenz de Cabezon. $(n-1)$-st koszul homology and the structure of monomial ideals, 2008.

[4] Eduardo Saenz de Cabezon. Combinatorial koszul homology: Computations and applications, 2008.

[5] Vesselin Gasharov, Irena Peeva, and Volkmar Welker. The lcm-lattice in monomial resolutions. *Math. Res. Lett.*, 6(5-6):521–532, 1999.

[6] Bjarke Hammersholt Roune. The slice algorithm for irreducible decomposition of monomial ideals. *J. Symbolic Comput.*, (4):358–381, 2009.

Efficient algorithms for the algebraic analysis of system reliability

Eduardo Sáenz de Cabezón[a] Henry P. Wynn[b]

[a] Universidad de la Rioja (Spain)
e-mail: eduardo.saenz-de-cabezon@unirioja.es
[b] London School of Economics(U.K.)
e-mail: H.Wynn@lse.ac.uk

The reliability of multi-state coherent systems can be effectively analysed using an algebraic approach based on computations on monomial ideals [2]. Every multi-state coherent system has a monomial ideal associated to it and the knowledge of its multigraded Betti numbers and/or a certain form of its multigraded Hilbert series (based on free resolutions) provides exact reliability identities and good bounds for the reliability of the corresponding system. In particular, the bounds obtained from the minimal free resolution of the monomial ideal are tightest among all bounds based on resolutions. They are typically tighter than the generalised Bonferroni-Fréchet inclusion-exclusion bounds which can be shown that correspond to the well known Taylor resolution.

However, the computation of minimal free resolutions of monomial ideals is a hard computational problem in general. Also, when the size of the ideal grows, the computation of any resolution is a difficult task for the algorithms available in the main computer algebra systems. An alternative approach is to compute directly the ranks of the modules in the resolution without computing the resolution itself (i.e. the corresponding differentials). For this we use Mayer-Vietoris trees [4] which provide an efficient algorithm that has shown good performance and is able to deal with ideals of a reasonable size. This capabilities make this algorithm suitable for the analysis of system reliability [6].

In the poster we show the algebraic background of the correspondence between monomial ideals and multi-state coherent systems. Then, the Mayer-Vietoris tree algorithm is presented together with the application to our problem. Finally, we apply our methods to some of the most relevant systems in reliability theory (cf. [3] for instance). The systems analyzed include different types of k-out-of-n systems, an important family of coherent systems which have been the object of considerable attention in the reliability literature and are at the centre of the study of scan statistics, used, for example, in gene association studies. We study basic k-out-of-n systems, consecutive k-out-of-n systems, weighted k-out-of-n and multistate k-out-of-n. Another important family of sytems we study are *series-parallel* systems, a main example of coherent systems broadly used in the design and analysis of networks and other industrial applications.

When applied to these and other systems, our algorithms lead to recurrence relationships for the (multigraded) Betti numbers of the corrsponding ideals, which can be solved to give generating functions and explicit formulas for

the numbers and for the identities and bounds. From there it is possible to derive asymptotic formulas for exact reliability and bounds under models such as independence as, for example, $n \to \infty$. Some of these results are known from probability theory but many are new and compare well with bounds derived by other methods [5, 6].For a range of other problems without a fixed structure, when there are no combinatorial formulas or asymptotics results, the computer algorithms still give fast accurate solutions for relatively large systems. In particular, we don't need to make strong assumptions on the probability models, which is the case of most other approaches.

Our algorithm has been implemented in the `C++` library `CoCoALib` [1] and shows good performance, in particular when the number of variables grows, which makes it suitable for this kind of application. Our algorithms provide the form of the Hilbert series that is needed for the analysis of the reliability of systems in comparatively high number of variables. We give some tables and diagrams showing the performance of our algorithm in different kinds of systems, comparing with the results in the literature, when available.

References

[1] CoCoATeam: CoCoALib: a `C++` library for doing Computations in Commutative Algebra, Available at `http://cocoa.dima.unige.it`

[2] B. Giglio and H.P. Wynn, *Monomial ideals and the Scarf complex for coherent systems in reliability theory*, Annals of Statistics, 32:1289-1311,2004

[3] W. Kuo and M.J. Zuo, *Optimal reliability modelling*, Wiley and Sons, New Jersey, 2003

[4] E. Sáenz-de-Cabezón, *Combinatorial Koszul homology: Computations and Applications*, PhD Thesis, Universidad de La Rioja, 2008

[5] E. Sáenz-de-Cabezón and H.P. Wynn, *Computational algebraic algorithms for the reliability of generalized k-out-of n and related systems*, ACA 2008, special session "Nonstandard Applications of Computer Algebra", E. Roanes-Lozano, M. Wester, S. Steinberg (organizers).

[6] E. Sáenz-de-Cabezón and H.P. Wynn, *Betti numbers and minimal free resolutions for multi-state system reliability bounds*, Journal of Symbolic Computation, 44:1311-1325, 2009

Approximate Factorization of Polynomials in $Z[x]$

Tateaki Sasaki[a] Yasutaka Ookura[b]

[a] Institute of Mathematics, University of Tsukuba, Tsukuba-shi, Ibaraki 305-8571, Japan
[b] Graduate School, University of Tsukuba, Tsukuba-shi, Ibaraki 305-8571, Japan

Today, approximate polynomial factorization is well known, but it is only for multivariate polynomials over fields such as **C** or **R**. To our knowledge, no paper has been published so far on approximate factorization of univariate polynomials over $\mathbb{Z}$. In this poster, we challenge to this new theme, present three algorithms, and show that the approximate factorization in $\mathbb{Z}[x]$ is feasible but the factorization is often unstable, posing many problems to us.

Let $F(x)$ be a given primitive square-free polynomial in $\mathbb{Z}[x]$. Let $G(x)$, $H(x)$ and $\Delta(x)$ in $\mathbb{Z}[x]$ satisfy

$$F(x) = G(x)H(x) + \Delta(x), \qquad \|\Delta\| \le \varepsilon \ll 1,$$

then $G(x)$ and $H(x)$ are called approximate factors of $F(x)$ of *tolerance* ε, where $\|F\|$ denotes the norm of F. The $\Delta(x)$ is called *perturbation*. Let the roots of $F(x)$, $G(x)$ and $H(x)$ be

$$F(x) = f_l(x-\alpha_1)\cdots(x-\alpha_l), \quad G(x) = g_m(x-\gamma_1)\cdots(x-\gamma_m), \quad H(x) = h_n(x-\eta_1)\cdots(x-\eta_n),$$

where $\gamma \to \hat{\alpha}_{i_k}$ $(k=1,\ldots,m)$ and $\eta \to \hat{\alpha}_{j_k}$ $(k=1,\ldots,n)$ as $\|\Delta\| \to 0$. Let $\alpha_i = \hat{\alpha}_i + \delta_i$ $(i=1,\ldots,l)$, where $\delta_i \to 0$ as $\|\Delta\| \to 0$. We call δ_i *perturbations* of α_i. We define perturbations of the coefficients

similarly. Let I and J be the following index sets: $I = \{i_1, \ldots, i_m\}$ and $J = \{j_1, \ldots, j_n\}$. Assuming that the perturbations $\delta_1, \ldots, \delta_l$ are sufficiently small, we want to determine the index sets I and J. In this poster, we consider only the case of $f_l = g_m h_n$.

Algorithm 1 Sum of Powers of Roots (SPR) Method
This method was suggested by [1], and used for exact factorization in $\mathbb{Z}[x]$ by [2]. Let

$$N_k = f_l^k \times (\alpha_{i_1}^k + \cdots + \alpha_{i_m}^k) \qquad (k = 1, \ldots, m).$$

Then $N_k \in \mathbb{Z}$ if $\|\Delta\| = 0$. Therefore, finding an index set I such that $N_1, \ldots, N_k$ are approximately integers, we will be able to obtain an approximate factor.

Proposition 3. *The perturbation of N_k is $O(k f_l^k \max\{(|\alpha_1^{k-1}\delta_1|, \ldots, |\alpha_m^{k-1}\delta_m|\})$.* □

The SPR method works reasonably for monic polynomials. For non-monic polynomials, it is almost useless, because the perturbation of N_k increases rapidly as k increases.

Algorithm 2 Factor-Differentiation Method (FDM)
This method uses the idea in [3]. Let $F_i(x) = (x - \alpha_i)$ $(i = 1, \cdots, l)$. We define *factor-differentiated polynomial* $\hat{F}_i(x)$ as $\hat{F}_i(x) = F(x)/F_i(x)$. Let $\hat{F}_I(x) = \hat{F}_{i_1}(x) + \cdots + \hat{F}_{i_m}(x)$ and $\hat{F}_J(x) = \hat{F}_{j_1}(x) + \cdots + \hat{F}_{j_n}(x)$. Then, for $\Delta = 0$, we have $\hat{F}_I(x) = G'(x)H(x) \in \mathbb{Z}[x]$ and $\hat{F}_J(x) = G(x)H'(x) \in \mathbb{Z}[x]$. Therefore, finding an index set I such that $\hat{F}_I$ is a polynomial with approximately integer coefficients, we will be able to obtain an approximate factor. (Actually, we use only FD-polynomials with real coefficients: if α_i is imaginary, we define $F_i(x)$ and $\hat{F}_i(x)$ as $F_i(x) = (x - \alpha_i)(x - \bar{\alpha}_i)$ and $\hat{F}_i = [F(x)/F_i(x)]F_i'(x)$, respectively.)

Proposition 4. *The perturbation of the x^k-term of $\hat{F}_I$ is $O(\sum_{i=0}^{k}((k-i+1)|g_{k-i+1}\delta_{Hi}| + |h_{k-i}\delta'_{Gi}|))$, where g_i and h_i are the coefficients of x^i-terms of G and H, respectively, and δ'_{Gi} and δ_{Hi} are perturbations of x^i-terms of $[g_m \Pi_{i\in I}(x - \alpha_i)]'$ and $[h_n \Pi_{j\in J}(x - \alpha_j)]$, respectively.* □

For non-monic polynomials, the FDM is much better than the SPR method, but it still suffers from the leading coefficient (or tail coefficient). If f_l is large then either g_m, h_n or both are large, making perturbations of many coefficients of $\hat{F}_I(x)$ (and $\hat{F}_J(x)$) large. Furthermore, in order to determine g_m (or h_n), we assign f_l as the leading coefficient of candidate factor, which also increases the perturbations of the coefficients.

Algorithm 3 Robust Ultimate Method (RUM)
Let LC be a set of all the positive integer divisors of f_l, and let $\tilde{G}(x) \stackrel{\text{def}}{=} g\Pi_{i\in I}F_i(x), g \in \text{LC}, 1 \le m \le [l/2]$. We check if $\tilde{G}(x)$ divides $F(x)$ approximately at tolerance ε for all the possible candidates $\tilde{G}(x)$, then we will be able to find an approximate factor of tolerance ε. This is the third algorithm. One may think that this method will find any approximate factor of tolerance ε, but it is not true. For some ill-conditioned polynomials (Wilkinson-type polynomials), $\tilde{G}(x)$ cannot give approximate factors of the minimal tolerances.

Time complexity of this algorithm is proportional to the cardinality of LC, so it seems to be slow but it is not so slow actually. In the actual implementation, we first check the second leading and the last coefficients of $\tilde{G}(x)$ whether they are approximately integers. This check discards most wrong candidates quickly.

We have implemented the above three algorithms and tested variously. In order to find the desired index set I, we employed the knapsack method; the lattice method is useless for our algorithms. We found the following points by the tests.

- The algorithms are applicable to polynomials of several ten degrees.
- The SPR method works for monic polynomials well, but it is useless for non-monic polynomials. The FDM is unstable but RUM is stable.
- Approximate factorization in $\mathbb{Z}[x]$ is pretty sensitive to ε: there is no approximate factor if ε is small, but there may be many factors if ε is made bigger.
- In actual implementation, we select factor candidates by using a cutoff parameter ε_{cut}, $\varepsilon_{\text{cut}} > \varepsilon$. This seems to be not a good strategy, but we have no better strategy now.

References

[1] T. Sasaki and M. Sasaki. A unified method for multivariate polynomial factorizations. *Japan J. Indust. Appl. Math.*, 10:21-39, 1993.

[2] M. van Hoeij. Factoring polynomials and the knapsack problem. *J. Number Theory*, 95:167-189, 2002.

[3] K. Belabas, M. van Hoeij, J. Klueners, and A. Steel. Factoring polynomials over global fields. Preprint, 2004; `http://arxiv.org/abs/math/0409510`

Computer algebraic efficiency of matrix polynomials for a system of integral equations

Nikta Shayanfar[a] Mahmoud Hadizadeh[a]

[a] Department of Mathematics, Faculty of science, K. N. Toosi University of Technology, P.O. Box: 16315−1618, Tehran, Iran
e-mail: `shayanfar@dena.kntu.ac.ir`
e-mail: `hadizadeh@kntu.ac.ir`

Many computer algebra systems include facilities for the solutions of the systems of ordinary differential and integral equations. Besides, classical areas of computer algebra are dominated by algorithms which are based on algebraic concepts and emphasize on using computer algebra systems for computing symbolic solutions.

Consider the algebra of all complex matrices, the univariate matrix polynomial is defined as follows:

$$A(\lambda) = \sum_{i=0}^{n} A_n \lambda^n,$$

where λ is a complex variable and $A_i,\ i = 0, \cdots, n$, are complex matrices with $detA_n \neq 0$. However, the connection between systems of differential equations and univariate matrix polynomials is well known [3]. The present paper employs the canonical form of matrix polynomials theory to investigate the Smith form and applies the obtained results to the system of integral equations.

Canonical forms of matrices are useful tools for classifying matrices, identifying their key properties and reducing complicated systems of equations to equivalent diagonal systems. The fact that they are very difficult to be computed, makes them particularly amenable to computer algebra. Indeed, working with matrix polynomials, one fundamental canonical form, the Smith form is defined as follows:

Every $n \times n$ matrix polynomial $A(\lambda)$, with $detA(\lambda) \neq 0$, admits a representation [1]:

$$A(\lambda) = E(\lambda)D(\lambda)F(\lambda),$$

where $E(\lambda)$ and $F(\lambda)$ are $n \times n$ matrix polynomials with constant nonzero determinants and the Smith form $D(\lambda) = diag[d_1(\lambda), \cdots, d_r(\lambda)]$ is a diagonal matrix with monic scalar polynomials $d_i(\lambda)$ such that $d_i(\lambda)$ is divisible by $d_{i-1}(\lambda)$; Definitions and characterization of Smith form is also given for the multivariate matrix polynomials expressed as the generalization of the scalar multivariate polynomials. Moreover, the bivariate matrix polynomials expressed in the homogeneous form is defined as follows [2]:

$$P(\alpha, \beta) = A_d\alpha^d + A_{d-1}\alpha^{d-1}\beta + \cdots + A_0\beta^d.$$

The most common application of this factorization involves solving the system of linear differential and difference equations. In this research, the main idea is to approximate the system of linear integral equations by a matrix polynomial equation. A method for constructive solution of the system is based on the Smith form of the corresponding matrix polynomials. First, we transform the integral operators of the system to a matrix polynomial $A(\lambda)$. In fact, the novel matrix polynomial is obtained if the variable is replaced by the integral operator. However, various kernels

in the system of integral equations lead to multivariate matrix polynomials. Then, diagonalisation $D(\lambda)$ of a matrix polynomial using elementary transformation should be obtained. Symbolic computer algebra relieve one from the tedious task of different mathematical operations, which are essential to obtain the Smith form.

The search for factorization acting on a class of multivariate matrix polynomials under consideration has been significantly useful in finding the solution of the system of integral equations as follows:

$$\mathbf{X}(t) + \int_a^t \mathbf{K}(t,s)\mathbf{X}(s)ds = \mathbf{Y}(t),$$

where

$$X(t) = [x_1(t), x_2(t), ..., x_n(t)]^T, \quad Y(t) = [y_1(t), y_2(t), ..., y_n(t)]^T,$$
$$K(t,s) = [K_{ij}(t,s)], \quad i,j = 1,2,...,n.$$

and $K(t,s)$ and $Y(t)$ are continuous given functions and $X(t)$ is the solution to be determined. The corresponding matrix polynomial equation to the above system can be considered as follows:

$$A(\lambda)X = Y.$$

Here, the variables in the matrix polynomial are governed by integral operators. Due to the highly advanced features of symbolic computations and the computational complexity of system solution, using canonical form, we split the integral equations system to a system of independent equations.

$$D(\lambda)U(t) = Z(t),$$

where $U(t) = F(\lambda)X(t)$ and $Z(t) = E^{-1}(\lambda)Y(t)$. So, the desired solution of the altered system can be solved easily by any classical methods and acting integral operations on these solutions will lead to the final solution of the system. Finally, an applicable model is included to show the validity and applicability of this approach in comparison to most numerical methods.

References

[1] I. Gohberg, P. Lancaster and L. Rodman, *Matrix Polynomials,* Academic press, New York, 1982.

[2] N. J. Higham and F. Tisseur, *More on pseudospectra for polynomial eigenvalue problems and applications in control theory,* Lin. Alg. Appl. 351-352, 435-453 (2002).

[3] P. Lancaster and M. Tismenetsky, *The Theory of Matrices with Applications*, Second Ed., Academic Press, 1985.

A New Method of Reducing Exact Computations to Obtain Exact Results

Kiyoshi Shirayanagi[a] Hiroshi Sekigawa[b]

[a] Department of Mathematical Sciences, Tokai University, Japan
e-mail: shirayan@tokai-u.jp
[b] NTT Communication Science Laboratories, Nippon Telegraph and Telephone Corporation, Japan
e-mail: sekigawa@theory.brl.ntt.co.jp

In this poster, we will propose a new method that reduces the number of exact computational steps needed for obtaining exact results, for a certain class of algebraic algorithms. This method is the floating-point interval method using zero rewriting and symbols. Zero rewriting, which is from stabilization techniques [1], rewrites an interval coefficient into the zero interval if the interval contains zero. Symbols are used to keep track of the execution path of the original algorithm with exact computations, so that the associated real coefficients can be computed by evaluating the symbols. The key point is that at each stage of zero rewriting, one checks to see if an interval that has been rewritten into zero is really zero by exploiting the associated symbol. This method mostly uses floating-point computations; the exact computations are only performed at the stage of zero rewriting and in the final evaluation to get the exact coefficients. Moreover, one does not need to check the correctness of the output.

Now let us describe more details. We restrict ourselves to the following class of algorithms:

- Input, intermediate, and output data are from the polynomial ring $R[x_1, \ldots, x_m]$, where R is a subring of the real numbers.
- Operations on data are polynomial functions over R.
- Predicates on data have *zero discontinuity*.

A predicate is said to have *zero discontinuity* if it has no discontinuous points or the only discontinuous point of the predicate is *zero* such as in the if statement "If $C = 0$ then ... else ...". Let us call the class of algorithms that satisfy the above three conditions, *algebraic algorithms with zero discontinuity*.

Given an algebraic algorithm with zero discontinuity, our proposed method is as follows:

R-to-IS Make each input coefficient a into the pair (*interval with symbol* or *IS* for short) $[[\tilde{a}, \alpha], Symbol_a]$, where $[\tilde{a}, \alpha]$ is an interval of a ($\tilde{a}$ is a floating-point approximation of a with a preselected precision and α is an error bound) and $Symbol_a$ is an indeterminate representing a.

IS Arithmetic Perform arithmetic between IS's:

$$[[A, \alpha], s] + [[B, \beta], t] = [[A, \alpha] + [B, \beta], \dot{+}(s, t)]$$

$$[[A, \alpha], s] - [[B, \beta], t] = [[A, \alpha] - [B, \beta], \dot{-}(s, t)]$$

$$[[A, \alpha], s] \times [[B, \beta], t] = [[A, \alpha] \times [B, \beta], \dot{\times}(s, t)]$$

Namely, for the interval parts, perform *interval arithmetic*, and for the symbol parts, just remember what arithmetic is done, by using *formal* symbols $\dot{+}$, $\dot{-}$, and $\dot{\times}$ for addition, subtraction, and multiplication.

Guaranteed Zero Rewriting For any intermediate IS $[[E, \epsilon], s]$, if $|E| \leq \epsilon$, then substitute the original input coefficient values for the symbol s to make the associated real number $r(s)$. If $r(s) = 0$, proceed to the next step; if otherwise, raise the precision and go back to **R**-to-IS.

IS-to-R Substitute the original input coefficient values for the respective symbols to evaluate the symbol part of the output.

Here, as a simple example of evaluating a symbol, let $s = \dot{\times}(t, \dot{+}(u, v))$ be the symbol part of an intermediate or output IS, where t, u and v were input symbols for $\sqrt{2}$, 3, and 4, respectively. Then by performing the substitution $t = \sqrt{2}$, $u = 3$, and $v = 4$ for s, one should obtain the result $r(s) = \sqrt{2} \times (3 + 4) = 7\sqrt{2}$.

We refer to the above method as the *ISCZ method* (the *IS method with Correct Zero rewriting*). The termination and correctness of this method can be proved by the theory of stabilizing algebraic algorithms [1]. Namely, if the given algorithm $\mathcal{A}$ terminates with an input I, then the ISCZ method for $\mathcal{A}$ always terminates in a finite number of steps and gives the same result as the output of $\mathcal{A}(I)$.

Using the ISCZ method, one can skip exact computations (evaluation of s) of any IS $[[E, \epsilon], s]$ and use only floating-point computations unless $|E| \leq \epsilon$. In other words, one can reduce the exact computations of nonzero coefficients. Therefore, the ISCZ method is effective when there are more cases where $|E| > \epsilon$ than those where $|E| \leq \epsilon$. Moreover, one does not need to check the correctness of the output.

However, the ISCZ method can obviously cause the symbol parts of IS's to grow. In this poster, we also will propose a way to circumvent the growth of the symbol parts.

We applied the ISCZ method to Buchberger's algorithm that computes Gröbner bases w.r.t. the lexicographic order in Maple on a computer. The experimental results indicated that the ISCZ method is more effective in the case of non-rational coefficients, especially the ones including transcendental constants, than in the case of rational coefficients. We will describe more details on the experiment in this poster.

In general, we believe that the ISCZ method is useful when the growth of intermediate coefficients is the main reason exact computation of the original algorithm is slow. In Buchberger's algorithm one often has huge intermediate coefficients even though the final coefficients are moderate in size. Investigation on other algorithms is a future work.

References

[1] K. Shirayanagi and M. Sweedler. A theory of stabilizing algebraic algorithms. Technical Report 95-28, Mathematical Sciences Institute, Cornell University, 1995. 92 pages.
`http://www.ss.u-tokai.ac.jp/~shirayan/msitr95-28.pdf`

Computing Gröbner Bases within Linear Algebra and Its Implementation

Akira Suzuki[a1]

[a] e-mail: `sakira@kobe-u.ac.jp`

The concept of Gröbner bases and the algorithm to compute them were introduced by Buchberger and the algorithm consists of computation of S-polynomials and one of monomial reductions. Since it has applications across a wide range of computer algebra, many optimizations have been studied and developed [6, 10, 7, 1]. In [4], Faugère introduced a new efficient algorithm F_4 to compute Gröbner bases by use of linear algebra in order to achieve simultaneous monomial reductions. A method to solve systems of algebraic equations by use of linear algebra were also studied by Lazard in [8]. Also by appropriate choices of term orders for given ideals, we may compute Gröbner basis for it efficiently. Mora-Robbiano [9] and Caboara [2] studied several method to find suitable term orders.

We introduce a new method to compute Gröbner bases of ideals on polynomial rings by use of sparse linear algebra. It implicitly processes both of the computations of S-polynomials and the one of monomial reductions in a single linear space simultaneously. Thus, with this method, we can expect to use a great variety of techniques including parallelism used by computation in linear algebras, in order to get Gröbner bases. Moreover our algorithm includes an indicator to choose an appropriate term order for an efficient computation of Gröbner basis for a given system of polynomials. The term order changes dynamically during the computation. If we need the Gröbner basis with respect to a given term order, we can use a method for change of order, e.g., Gröbner walk [3], FGLM [5], and Hilbert driven [12].

We choose PARI/GP[2] to implement the algorithms in this paper in order to demonstrate that it is not difficult to implement Gröbner bases computation to a system which has neither S-polynomials nor monomial reductions. You can download the file from our page. [3]

This file has three main routines, (1) `groebner_basis_la`, (2) `groebner_basis_la_opt`, and (3) `groebner_basis_la_mat`. The first one is a naive implementation of our algorithm, the second is optimized version. Furthermore, we give the experimental version (3) omitting routine for computation of row echelon form.

In these main algorithms input polynomials are converted to vectors in a direct product of $\mathbb{Q}$ just before of computation, the computed vectors are converted to output polynomials at the end of the routines, and most of all computations are processed within linear algebra.

[1]This work was supported by KAKENHI (20500013)., Kobe University

[2]`http://pari.math.u-bordeaux.fr/`

[3]`http://kurt.scitec.kobe-u.ac.jp/~sakira/GBwithinLA/`

We give a tiny computational example as follows:
For an example, when we put `groebner_basis_la_opt([x^2+y^2+z^2-20, x+y-5, x*y*z-3])`,
it outputs `[[8*z^3+40*z-48, 2*y^2-10*y+(z^2+5), x+(y-5)], [x,y,z], [6, 3, 2]]` which means $\{z^3 + 5z - 6, 2y^2 - 10y + z^2 + 5, x + y - 5\}$ is the reduced Gröbner basis of $\langle x^2 + y^2 + z^2 - 20, x + y - 5, xyz - 3\rangle_{\mathbb{Q}[x,y,z]}$ with respect to the term-order weighted by $\{x : 6, y : 3, z : 2\}$. In the following table, we give a timing data of these implementations though they are not faster than existing implementations to compute Gröbner bases. The unit of time in the table is the second. (Mac OS X 10.5.6, CPU 2.8GHz Xeon, Memory 22GB, GP/PARI 2.3.4 with x86-64/GMP-4.2.3)

polynomial system	(1) no opt.	(2) opt.	(3) no echelon
$\{xy + z - xy, x^2 - z, x^3 - x^2yz\}$	10.5	**1**.5	10.6
$\{xy + z - xz, x^2 - z, 2x^3 - x^2yz - 1\}$	>1 hour	**1**.6	11.5
$\{5x^3 - 7yz, 11y^2 - 101z, x + y - 65537z\}$	>1 hour	9.3	**0**.8
$\{x^2 + y^2 + z^2 - 20, x + y - 5, xyz - 3\}$	>1 hour	**3**.7	12.1

References

[1] Bosma, W., Cannon, J., and Playoust, C. (1997) *The Magma algebra system. I. The user language.* J. Symb. Comput., **2**4(3-4), 235-265.

[2] Caboara, M. (1993) *A dynamic algorithm for Gröbner basis computation* Proc. ISSAC '93, 275–283.

[3] Collart, S., Kalkbrenner, M., and Mall, D. (1997) *Converting Bases with the Gröbner Walk.* J. Symb. Comput. **2**4(3–4), 465–469.

[4] Faugère, J.C. (1999) *A new efficient algorithm for computing Gröbner bases (F_4).* J. Pure and Applied Algebra, **1**39(1–3), 61–88.

[5] Faugére, J.C., Gianni, P.M., Lazard, D., and Mora, T. (1993) *Efficient Computation of Zero Dimensional Gröbner Bases by Change of Ordering.* J. Symb. Comput. **1**6(4), 329-344.

[6] Gebauer, R., Möller, H.M. (1988) *On an installation of Buchberger's algorithm.* J. Symb. Comput. **6**(2-3), 275–286.

[7] Greuel, G.M., Pfister, G (2006) *SINGULAR and Applications.* Jahresbericht der DMV **1**08, 167-196

[8] Lazard, D. (1983) *Gröbner-Bases, Gaussian elimination and resolution of systems of algebraic equations.* EUROCAL 1983, 146-156.

[9] Mora, T. and Robbiano, L. (1988) *The Gröbner fan of an ideal.* J. Symb. Comput. **6**(2-3) 183–208.

[10] Noro, M. and Takeshima, T. (1992). *Risa/Asir – A Computer Algebra System.* Proc. ISSAC '92, 387–396.

[11] Traverso, C. (1988) *Gröbner trace algorithms.* Proc. ISSAC f88 (LNCS 358), 125-138.

[12] Traverso, C. (1996) *Hilbert functions and the Buchbeger algorithm.* J. Symb. Comput. **2**2(4), 355–376.