A MODULAR METHOD FOR COMPUTING GRÖBNER BASES*

Jennifer de Kleine
jkleine@cecm.sfu.ca

Michael Monagan
monagan@cecm.sfu.ca

Centre for Experimental and Constructive Mathematics
Simon Fraser University
Burnaby, British Columbia, V5A 1S6 Canada

Keywords: Computing Gröbner Bases, Modular Algorithms, Algebraic Geometry, Solving Equations, the External Function Interface.
Intended Audience: Teachers and Users of Algebraic Geometry, Maple Developers.

Abstract

Buchberger's algorithm computes a Gröbner basis for a system of polynomials over a field. When working over the field of rational numbers $\mathbb{Q}$, the coefficients blow-up in the intermediate computations which slows Buchberger's algorithm down dramatically. The goal of this project is to design and implement a data structure that supports an efficient implementation of Buchberger's algorithm using the modular method. The main idea is to compute a Gröbner basis modulo batches of primes and then to reconstruct the Gröbner basis over the rationals using Chinese remaindering and rational reconstruction. The reason to compute modulo batches of primes is twofold. Firstly, to speed up the overall computation, by amortizing the monomial overhead across the batch of primes; and secondly to identify with high probability primes which cannot be used.

We have implemented the ideas in Maple V Release 7. We represent the coefficients of a polynomial with t terms in n variables modulo b primes as a b by t matrix of floating point numbers. We represent the monomials as an n by t matrix of integers, where each column stores the exponent vector of a monomial. We have also implemented the critical parts of the algorithm in C using the external function interface provided by Maple. The timing results obtained confirm the power of the modular method and show that working with batches of primes yields a significant improvement.

1 Introduction

Let $F = \{f_1, f_2, ..., f_s\} \subset k[x_1, x_2, ..., x_n]$ be a system of s polynomials in n unknowns over a field k. In 1965 Buchberger [3] defined a Gröbner basis G for the ideal $I = \langle f_1, f_2, ..., f_s\rangle$ and gave an algorithm, now commonly called "Buchberger's algorithm" for computing G. Roughly speaking, a Gröbner basis is the analogue of row Echelon form for a linear system of equations and "Buchberger's algorithm" is the analogue of Gaussian elimination. Here we give some definitions and examples. Details can be found in the undergraduate text of Cox et. al. [8].

Definition 1*(Gröbner basis): Let I be an ideal in $k[x_1, ..., x_n]$ and let $\succ$ be any monomial or term ordering on the monomials in $x_1, ..., x_n$. A basis $G = \{g_1, ..., g_t\}$ for I is said to be a* **Gröbner basis** *for I w.r.t. $\succ$ if $f \in I \iff Rem_{\succ}(f, G) = 0$.*

*This work was supported by the MITACS NCE of Canada

Here $Rem_{\succ}(f, G)$ denotes the remainder on division of the polynomial f by the set of polynomials G under the monomial ordering $\succ$. Gröbner bases have many applications. Since a Gröbner basis G for an ideal I generated by F is a basis for I, it follows that any solution of F is a solution of G and vice versa. Thus one of the main applications of Gröbner bases is describing the solutions of a system of polynomial equations. Consider the system

$$F = \{x^2 + y^2 + z^2 + 1, x^2 + yz - 1, y^2 + zx - 1\}.$$

A Gröbner basis for $I = \langle x^2+y^2+z^2+1, x^2+yz-1, y^2+zx-1\rangle$ under the lexicographic monomial ordering ($\succ_{lex}$) with $x > y > z$ is

$$G = \{6x + 3z^3 + 5z, 6y + 3z^3 + 5z, 3z^4 + 8z^2 + 9\}.$$

By looking at G it is easy to see that this system has four solutions, which can be solved for by first solving the equation in z only and then back-substituting to solve for y and x. Other information about the ideal can be easily found from the Gröbner basis. For example, since the first two polynomials in G are linear in x and y respectively, and since the third polynomial $3z^4 + 8z^2 + 9$ is irreducible over the rationals (apply factor in Maple to verify this), it follows that I is maximal hence the quotient ring $k[x_1, ..., x_n]/I$ is a field. We can also determine easily whether a given polynomial f is in I. This follows directly from the definition. For example consider $f = x - y$. Looking at F, because $Rem_{\succ_{lex}}(f, F) = x - y$, we cannot tell whether or not f lies in I. However, given the Gröbner basis G we find that $Rem_{\succ_{lex}}(f, G) = 0$, verifying that $f \in I$.

The definition we gave for Gröbner bases is not constructive; it does not give us a practical means to determine if a given basis $G = \{g_1, ..., g_t\}$ for an ideal $I \subset k[x_1, ..., x_n]$ is a Gröbner basis for I because I is an infinite set of polynomials. To do this we need another characterization for Gröbner bases which is based on the idea of a "Syzergy polynomial" or "S-polynomial" for short. Given two polynomials, f and g, the "S-polynomial" of f and g, denoted $S_{\succ}(f, g)$, is the linear combination of f and g which causes the leading terms under $\succ$ to cancel.

Example: Let $f = 2xy^3z^2 + 1$ and $g = 3yz^3 + z$. Under $\succ_{lex}$ with $x > y > z$

$$S(f, g) = (3z)f - (2xy^2)g = [6xy^3z^3 + 3z] - [6xy^3z^3 + 2xy^2z] = -2xy^2z + 3z.$$

Theorem (Buchberger's S-pair criterion [4]): Let I and $\succ$ be as in definition 1.
A basis $G = \{g_1, ..., g_t\}$ for I is a Gröbner basis for I w.r.t. $\succ$
$\iff Rem_{\succ}(S(g_i, g_j), G) = 0 \; \forall \, i \neq j, \, i, j \in \{1, ..., t\}$.

Not only does this characterization provide a means for determining whether a given basis is a Gröbner basis it also leads to a method for computing Gröbner bases, namely Buchberger's algorithm:

INPUT: $F = \{f_1, f_2, ..., f_s\} \subset k[x_1, x_2, ..., x_n]$ and $\succ$ a monomial ordering.
OUTPUT: $G = \{g_1, g_2, ..., g_t\}$ a Gröbner basis for $\langle f_1, ..., f_s\rangle$ wrt $\succ$.
$G \leftarrow F$
do
  $\overline{G} \leftarrow G$
  for each pair $\{f, g\} \subset \overline{G}$, $f \neq g$ do
    $r \leftarrow \mathrm{Rem}_{\succ}(S_{\succ}(f, g), \overline{G})$
    if $r \neq 0$ then $G \leftarrow G \cup \{r\}$
while $G \neq \overline{G}$
RETURN G

A difficulty that arises in computing Gröbner bases over the rationals is that the coefficients of the intermediate polynomials may become very large. This results in much time being spent on rational arithmetic. Many people have investigated how to speed up this basic algorithm, see [5, 6, 7, 10]. One way to deal with this problem is to apply modular methods. The simplest way to do this is to compute the Gröbner basis modulo primes $p_1, p_2, ...$ and then apply the Chinese remainder theorem and rational reconstruction, [12], to recover the Gröbner basis G over $\mathbb{Q}$. It may happen that the Gröbner basis we compute modulo a given prime p is wrong, that is, not equal to G mod p. Such bad primes can be detected once we have sufficiently many good primes by inspecting the leading monomials of the modular Gröbner bases. A number of people have looked at modular methods for computing Gröbner bases, for example, see [11, 1].

We have observed that the bottleneck in computing Gröbner bases over $\mathbb{Z}_p$, p a machine prime, is the division algorithm, and when working in $\mathbb{Z}_p[x_1, ..., x_n]$ most of the time is spent doing arithmetic and memory management of monomials instead of coefficient arithmetic in $\mathbb{Z}_p$. Our idea to address this problem is to compute using batches of primes, that is, to work over $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}$, doing coefficient operations simultaneously on b primes to amortize the monomial cost over the cost of arithmetic in $\mathbb{Z}_p$. This is a better solution than trying to use sparse linear algebra to get back efficiency. Another advantage of this approach is that we will identify bad primes immediately.

Our approach involved the design of a data structure that supports an efficient modular implementation of Buchberger's algorithm and a prototype implementation of this method in Maple V Release 5. A detailed report of this work can be found in [9]. The bottleneck in the prototype Maple implementation is the monomial arithmetic and vector operations in $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}$. We have made further improvements by implementing these critical parts in C by using the external interface provided in Maple 7.

This article is organized as follows. In section 2 we discuss the data structure that we are using in our current implementation. In section 3 we present timing results on three moderately hard problems, Trinks, Cubics and Rose. We conclude with a discussion of the timing results obtained, a comparison with Maple's gbasis and future work in section 4. The results confirm the power of the modular method and show that working with batches of primes yields a significant improvement.

2 Data Structure

The data structure used is a sparse distributive ordered representation for multivariate polynomials over $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}$. The purpose of this data structure is to provide efficient computation of Gröbner bases of multivariate polynomials over $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}$. In particular, it is designed to be used with the external C interface provided in Maple 7.

We are using b by t matrices of hardware floating point numbers to represent the coefficients of a polynomial with t terms in n variables modulo $p_1, p_2, ..., p_b$; i.e., each column of the matrix encodes a coefficient mod $p_1, p_2, ..., p_b$. The choice of floating point numbers is firstly for speed, floating point arithmetic is faster than integer arithmetic because it has better hardware support. Secondly, double precision floats allow us to use 8 decimal digit primes rather than 4.7 digit primes, which is what we would get using integers. We represent the monomials as an n by t matrix of integers, where each column stores the exponent vector of a monomial.

Given a polynomial $f = c_1 x^{\alpha_1} + c_2 x^{\alpha_2} + ... + c_t x^{\alpha_t} \in \mathbb{Z}[x_1, ..., x_n]$, a batch of primes $[p_1, ..., p_b]$ and a monomial ordering $\succ$, this polynomial is mapped to $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}[x_1, ..., x_n]$ and stored as shown:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1t} \\ c_{21} & c_{22} & \cdots & c_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ c_{b1} & c_{b2} & \cdots & c_{bt} \end{bmatrix} M = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1t} \\ m_{21} & m_{22} & \cdots & m_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nt} \end{bmatrix}.$$

Here, $c_{ij} = c_j \bmod p_i$, $1 \leq i \leq b$, $1 \leq j \leq t$, and $m_j = x_1^{m_{1j}} x_2^{m_{2j}} \cdots x_n^{m_{nj}} = x^{\alpha_k}$ for some k, $1 \leq k \leq t$, for $1 \leq j \leq t$ such that $m_1 \succ m_2 \succ \cdots \succ m_t$. Hence, m_1 is the leading monomial of f.

For example, consider the polynomial $f = 11x^2y - 7y^2 + 13z \in \mathbb{Z}[x, y, z]$. For the batch of primes $[2, 3, 5, 7]$ and the ordering $\succ_{lex}$ with $x > y > z$, f is stored as follows:

$$C = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ 1 & 3 & 3 \\ 4 & 0 & 6 \end{bmatrix} M = \begin{bmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

3 Results

This method was implemented in Maple V Release 7 as described previously and tested on various problems, varying the batch size and using prime sizes of 8 decimal digits. For each test problem, the batch size versus the total CPU time used in seconds is plotted. What we hope to see is the following: as we increase the batch size, the CPU time should drop. The problems were run on an Intel Celeron 500 Mhz system with 192 Mb of RAM. The test problems run include the following.

3.1 Trinks

This problem of Trinks, in $\mathbb{Q}[b, s, p, t, z, w]$ is known to be hard for most lex orderings. Trinks under $\succ_{lex}$ with $b > s > p > t > z > w$ is a hard problem. The largest coefficient in the Gröbner basis of this system under $\succ_{lex}$, $\succ_{grevlex}$ and $\succ_{grlex}$ has 161, 38 and 40 decimal digits, respectively. The results of running the modular method on this problem under $\succ_{lex}$, $\succ_{grevlex}$ and $\succ_{grlex}$ are plotted in figures 1, 2 and 3, respectively.

$$\begin{gathered}\text{Trinks} = [45p + 35s - 165b - 36,\\ 35p + 40z + 25t - 27s,\\ 15w + 25ps + 30z - 18t - 165b^2,\\ -9w + 15pt + 20zs,\\ wp + 2zt - 11b^3,\\ 99w - 11sb + 3b^2]\end{gathered}$$

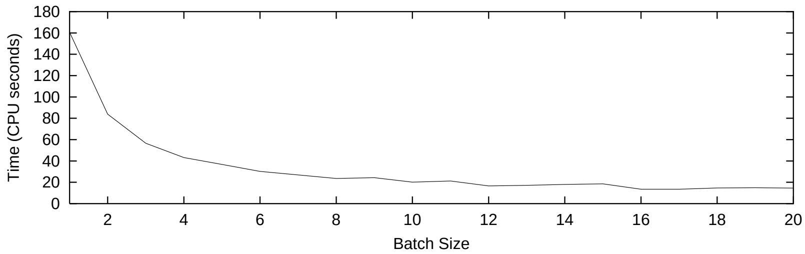

Figure 1: Trinks, $\succ_{lex}$, $b > s > p > t > z > w$

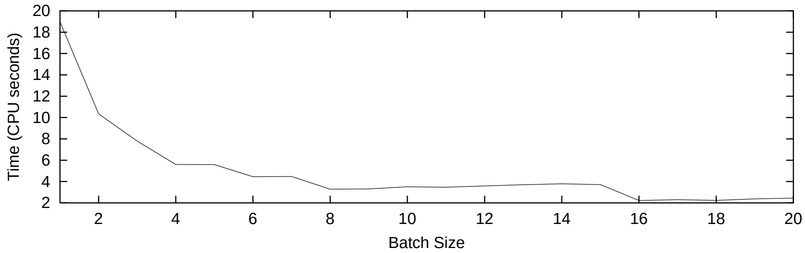

Figure 2: Trinks, $\succ_{grevlex}$, $b > s > p > t > z > w$

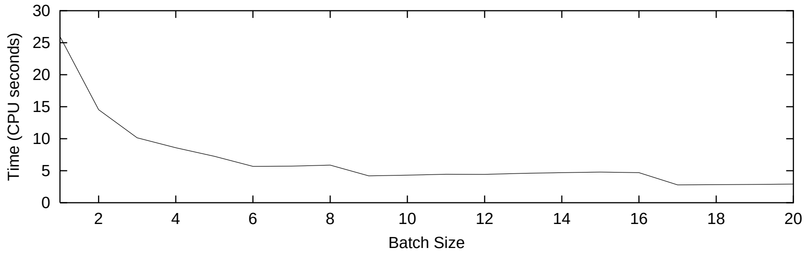

Figure 3: Trinks, $\succ_{grlex}$, $b > s > p > t > z > w$

3.2 Randomly Generated Cubics

This system contains 3 randomly generated degree 3 polynomials in $\mathbb{Q}[x, y, z]$. The largest coefficient in the Gröbner basis of this system under $\succ_{lex}$, $\succ_{grevlex}$ and $\succ_{grlex}$ has 667, 62 and 61 decimal digits, respectively. The results of running the modular method on this problem under $\succ_{lex}$, $\succ_{grevlex}$ and $\succ_{grlex}$ are plotted in figures 4, 5, and 6, respectively.

$$\begin{aligned}\text{Cubics} = [&79z + 56yz + 49yz + 63x + 57y - 59x,\\ &77y^2z + 66y^3 + 54xy - 5xyz + 99x - 61,\\ &-47x^2y - 91y - 47xyz - 61xz + 41xy^2 - 58x]\end{aligned}$$

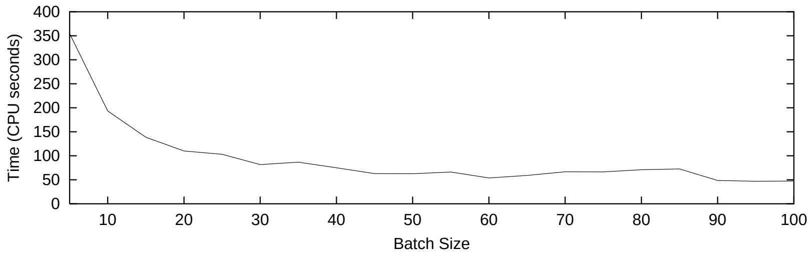

Figure 4: Cubics, $\succ_{lex}$, $x > y > z$

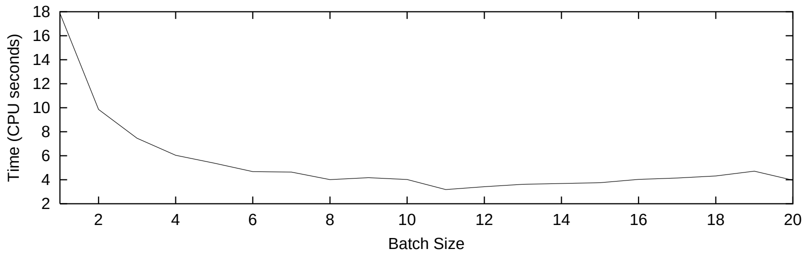

Figure 5: Cubics, $\succ_{grevlex}$, $x > y > z$

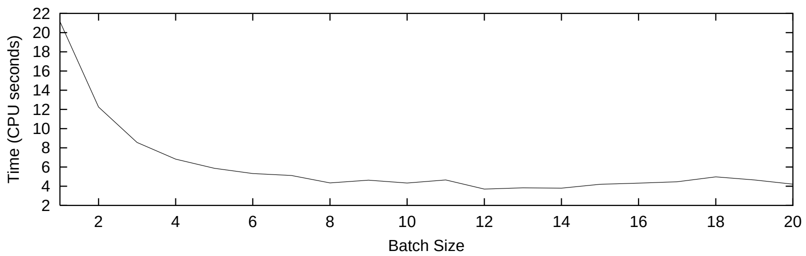

Figure 6: Cubics, $\succ_{grlex}$, $x > y > z$

3.3 Rose

This system in $\mathbb{Q}[x, y, z]$ was taken from the Posso test suite [2]. The largest coefficient in the Gröbner basis of this system under $\succ_{lex}$, $\succ_{grevlex}$ and $\succ_{grlex}$ has 903, 43 and 43 decimal digits, respectively. The results of running the modular method on this problem under $\succ_{lex}$, $\succ_{grevlex}$ and $\succ_{grlex}$ are plotted in figures 7, 8 and 9, respectively.

$$\text{Rose} = [y^4 - \tfrac{20}{7}x^2,$$
$$x^2z^4 + \tfrac{7}{10}xz^4 + \tfrac{7}{48}z^4 - \tfrac{50}{27}x^2 - \tfrac{35}{27}x - \tfrac{49}{216},$$
$$\tfrac{3}{5}x^6y^2z + x^5y^3 + \tfrac{3}{7}x^5y^2z + \tfrac{7}{5}x^4y^3 - \tfrac{7}{20}x^4yz^2 - \tfrac{3}{20}x^4z^3 + \tfrac{609}{1000}x^3y^3 + \tfrac{63}{200}x^3y^2z - \tfrac{77}{125}x^3yz^2 - \tfrac{21}{50}x^3z^3 +$$
$$\tfrac{49}{1250}x^2y^3 + \tfrac{147}{2000}x^2y^2z - \tfrac{23863}{60000}x^2yz^2 - \tfrac{91}{400}x^2z^3 - \tfrac{27391}{800000}xy^3 + \tfrac{4137}{800000}xy^2z - \tfrac{1078}{9375}xyz^2 - \tfrac{5887}{200000}xz^3 - \tfrac{1029}{160000}y^3 -$$
$$\tfrac{24353}{1920000}yz^2 - \tfrac{343}{128000}z^3]$$

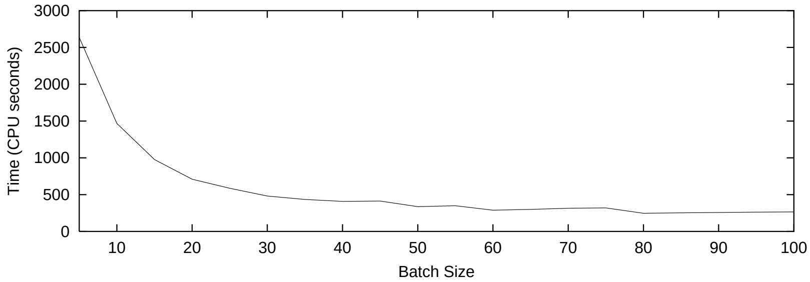

Figure 7: Rose, $\succ_{lex}$, $x > y > z > t > u$

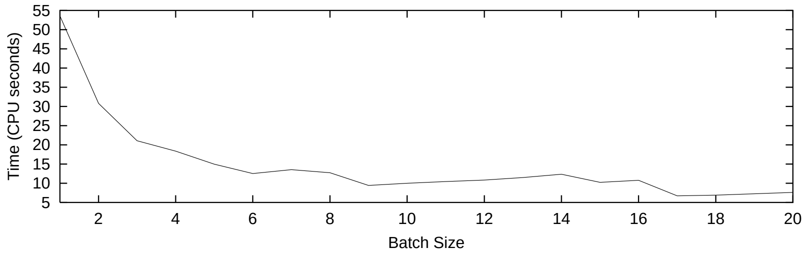

Figure 8: Rose, $\succ_{grevlex}$, $x > y > z > t > u$

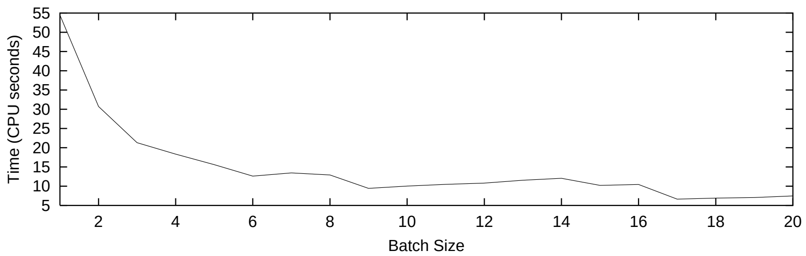

Figure 9: Rose, $\succ_{grlex}$, $x > y > z > t > u$

4 Conclusion and Future Work

We sought to improve the efficiency of the modular method. The bottleneck is the monomial overhead. The main improvement is to compute the Gröbner basis modulo batches of primes. The timing results in section 3 show that as we increase the batch size b the computing time drops significantly.

The other main improvement was to make use of the external function interface to speed up the implementation of the critical parts of the algorithm. We ran the gbasis procedure in Maple V Release 4 and in rmaple, which have a better implementation under the lexicographic ordering than Release 5, 6 or 7, on the test problems. We compare the times achieved by Maple together with the best times achieved by our modular method for the three test problems under the lexicographic ordering. The performance of the modular method shows a huge improvement over the Maple implementations. We summarize the results in the following table, the times given are in CPU seconds.

problem	gbasis (Maple V R4)	gbasis (rmaple)	modular method
Trinks	killed at 61944.91	killed at 53562.10	6.65 (batch size 50)
Cubics	8945.25	9496.46	36.56 (batch size 200)
Rose	41674.16	22204.75	169.17 (batch size 300)

Profile results indicate that the bottleneck in the current implementation is the division algorithm. We are currently implementing the full division algorithm externally in C using the external interface provided in Maple 7. We expect to get further gains in doing so.

We are also thinking about how to generalize the implementation so that it will work for parameters, e.g. the polynomial ring $k(a_1,...,a_m)[x_1,...,x_n]$. In principal we can evaluate out the parameters $a_1,...,a_m$ and recover the rational functions in them again using Chinese remaindering and rational reconstruction.

References

[1] Elizabeth Arnold. *Modular Methods for Computing Groebner Bases.* Poster Session, International Symposium on Symbolic and Algebraic Computation, 2001.

[2] D. Bini and B. Mourrain. *Polynomials test suite.* 1996.
⟨ http://www-sop.inria.fr/saga/POL/BASE/3.posso/rose.dir/index.html ⟩ (Feb. 2002).

[3] B. Buchberger. *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal (German).* Ph.D. Thesis, Univ. of Innsbruck, Math. Inst., 1965.

[4] B. Buchberger. *A Theoretical Basis for the Reduction of Polynomials to Canonical Forms.* ACM SIGSAM Bull., 10(3), 1976, pp. 19-29.

[5] B. Buchberger. *A Criterion for Detecting Unnecessary Reductions in the Construction of Gröbner Bases.* Proc. EUROSAM '79, Lecture Notes in Computer Science 72, ed. W. Ng, Springer-Verlag, 1979.

[6] B. Buchberger and F. Winkler. *Miscellaneous Results on the Construction of Gröbner Bases for Polynomial Ideals.* Tech. Rep. 137, University of Linz, Math Inst., 1979.

[7] B. Buchberger. *Groebner bases: an algorithmic method in polynomial ideal theory.* Multidimensional Systems Theory, edited by N. K. Bose, D. Reidel Publishing Company, Dordrecht, pp. 184-232, 1985.

[8] David Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms - An Introduction to Computational Algebraic Geometry and Commutative Algebra.*(2nd ed). Springer-Verlag, New York, 1997.

[9] Jennifer de Kleine. *A Modular Design and Implementation of Buchberger's Algorithm.* MSc Project Report, Department of Computing Science, Simon Fraser University, 2001.

[10] Alessandro Giovini, Teo Mora, Gianfranco Niesi, Lorenzo Robbiano and Carlo Traverso. *"One sugar cube, please" OR Selection strategies in the Buchberger algorithm.* Association for Computing Machinery, 1991.

[11] Carlo Traverso. *Gröbner Trace Algorithms.* International Symposium on Symbolic and Algebraic Computation, pp. 125-138, 1988.

[12] Wang, P., Guy, M. J. T., and Davenport, J. H. p-adic reconstruction of rational numbers. *SIGSAM Bulletin 16*, 2 (1982).