A MODULAR DESIGN AND IMPLEMENTATION OF BUCHBERGER'S ALGORITHM

Jennifer de Kleine
dekleine@cs.sfu.ca

Michael Monagan
monagan@cecm.sfu.ca

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

1 Introduction

Buchberger's algorithm is an algorithm to compute a Gröbner basis for a system of polynomials in n unknowns over a field k. Bruno Buchberger both defined Gröbner bases and designed Buchberger's algorithm in 1965 [6]. Here is Buchberger's algorithm.

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

Buchberger's algorithm can be greatly improved upon. One improvement lies in the order in which the pairs of polynomials in the basis are selected for processing. Two such selection strategies are the "normal" selection strategy proposed by Buchberger [9], in which the "small" polynomials under the monomial ordering are considered first; and the "sugar" selection strategy, a refinement of the former, discussed by Giovani et al. in [3]. Another improvement is to reduce the number of S-polynomials which must be considered, hence reducing the number of divisions which must be performed. Two criterion were proposed by Buchberger [7, 8] which provide an inexpensive check to identify certain pairs of polynomials whose S-polynomial reduces to zero on division by the current basis.

One 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. One way to deal with this problem is to apply modular methods.

A number of people have looked at modular methods for computing Gröbner bases. Elizabeth Arnold presented a modular method for computing Gröbner bases in [5]. Her method uses Hensel lifting and the theory of Farey rationals. Once a candidate for the correct Gröbner basis is found it is checked to see if it is in fact the correct result. This check however, involves many divisions which can be quite expensive, hence a probabilistic approach will be more practical.

It has been 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 work involved is in the monomial arithmetic. This motivates our approach, to compute in batches of primes. That is, to work over $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}$, doing polynomial operations simultaneously on b primes to amortize the monomial cost over the cost of arithmetic in $\mathbb{Z}_p$.

An overview of the algorithm is as follows. Given a list of polynomials $[f_1, ..., f_s]$ in $\mathbb{Z}[x_1, ..., x_n]$ for which we want to compute a Gröbner basis, we first reduce the polynomials mod $p_1, ..., p_b$, for some primes $p_1, ..., p_b$ such that $p_i \nmid \mathrm{LC}(f_j)$ for $i = 1, .., b$, $j = 1, ..., s$. This gives us $G = [g_1, ..., g_s]$ in $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}[x_1, ..., x_n]$. Then we compute a reduced Gröbner basis of G using Buchberger's algorithm working over $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}$. The Gröbner basis over $\mathbb{Q}$ is then reconstructed by first applying the Chinese remainder algorithm to get a solution over $\mathbb{Z}_{p_1 \cdot ... \cdot p_b}$ and then applying rational reconstruction to get the final answer over $\mathbb{Q}$ [2]. If the $p_1 \cdot ... \cdot p_b$ is not large enough for the rational reconstruction to succeed then we get a new batch of primes, apply Buchberger's algorithm again to get another reduced Gröbner basis. Now, after applying the Chinese remainder algorithm to this batch we then apply Chinese remaindering to combine this result with the previous one. Once rational reconstruction succeeds we check the number of digits of agreement. If $\prod p_i$ is sufficiently larger than the largest numerator or denominator appearing in the coefficients, then we are done, otherwise, we get a new batch of primes and continue.

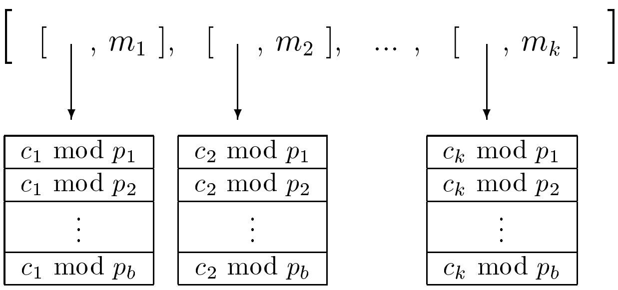
Our approach involved the design of a data structure that supports an efficient modular implementation of Buchberger's algorithm and a prototype implementation of this method in Maple V Release 5. The code is available online at http://www.cecm.sfu.ca/~jkleine.

This article is organized as follows. In section 2 the design of the data structure, including the monomial encodings and polynomial operations, is discussed. In section 3 you will find the implementation details of the modular method, including Chinese remaindering, rational reconstruction and the termination condition. In Section 4 the experimental results are given and in section 5 we conclude with a discussion of the timing results obtained. The results indicate great potential in this method.

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 for computing Gröbner bases of multivariate polynomials over $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}$. In particular, it is designed to be used in the modular implementation of Buchberger's algorithm mentioned previously.

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



Here, the m_i are the monomials of f in decreasing order under the term ordering $>$. That is, $m_i = x^{\alpha_j}$ for some $1 \leq j \leq k$ where $i < l \Rightarrow m_i > m_l$. Hence, m_1 is the leading monomial of f.

For example, consider the polynomial $f = 11x^2y - 7y^2 + 13z$. For the batch of primes $[2, 3, 5]$ and the lexicographic monomial ordering ($>_{lex}$), f is stored as follows.

$$\Big[\, [\, [\ 1,\ 2,\ 1\ ],\ x^2y\ ],\ [\, [\ 1,\ 2,\ 3\ ],\ y^2\ ],\ [\, [\ 1,\ 1,\ 3\ ],\ z\ ] \,\Big]$$

One representation for monomials is to store the exponent vectors, e.g. x^2y is represented by $[2, 1, 0]$. This requires $O(n)$ storage. To decrease the space requirements and to speed up the monomial processing we encode the monomials as non-negative integers, as is done in the computer algebra system Macaulay [4].

For the graded lexicographic ($>_{grlex}$) and graded reverse lexicographic ($>_{grevlex}$) monomial orderings the monomials are encoded as non-negative integers as discussed in the following section. For example, f under $>_{grevlex}$ is stored as follows. (see table 1)

$$\Big[\, [\, [\ 1,\ 2,\ 1\ ],\ 18\ ],\ [\, [\ 1,\ 2,\ 3\ ],\ 7\ ],\ [\, [\ 1,\ 1,\ 3\ ],\ 1\ ] \,\Big]$$

2.1 Monomial Encodings

For graded monomial orderings, there is a one-to-one correspondence between the non-negative integers and the monomials in n variables, or more specifically the exponent vectors of the monomials.

The one-to-one correspondence is defined as follows. Collect all monomials and order them in increasing order under some monomial ordering and then map the exponent vectors of the monomials to the integers, starting from 0. Thus given some monomial, the corresponding integer is its index into the given ordering. As well, given a non-negative integer m, the corresponding monomial is the m'th monomial under this ordering of monomials in n variables.

Consider the monomials in three variables, $[x, y, z]$. Under the graded reverse lexicographic ordering with $x > y > z$ we have the correspondence shown in table 1.

monomial	exponent vector	encoding
1	$[0,0,0]$	0
z	$[0,0,1]$	1
y	$[0,1,0]$	2
x	$[1,0,0]$	3
z^2	$[0,0,2]$	4
yz	$[0,1,1]$	5
xz	$[1,0,1]$	6
y^2	$[0,2,0]$	7
xy	$[1,1,0]$	8
x^2	$[2,0,0]$	9
z^3	$[0,0,3]$	10
yz^2	$[0,1,2]$	11
xz^2	$[1,0,2]$	12
y^2z	$[0,2,1]$	13
xyz	$[1,1,1]$	14
x^2z	$[2,0,1]$	15
y^3	$[0,3,0]$	16
xy^2	$[1,2,0]$	17
x^2y	$[2,1,0]$	18
x^3	$[3,0,0]$	19
$\vdots$	$\vdots$	$\vdots$

Table 1: Encoding of monomials in $[x, y, z]$ under $>_{grevlex}$

Given a monomial $m = x_1^{d_1} \cdot x_2^{d_2} \cdot ... \cdot x_n^{d_n}$, we have corresponding exponent vector $[d_1, ..., d_n]$.

Under the **graded lexicographic ordering** we have the following mappings. The encoding of the monomial m is the integer:

$$\binom{n + d_1 + ... + d_n}{n} - \sum_{i=1}^{n-1} \binom{i + d_n + d_{n-1} + ... + d_{n-(i-1)} - 1}{i} - 1.$$

Given a non-negative integer p, the corresponding monomial in n variables with exponent vector r is computed as follows:

1. Compute
$$\max\{\, d \mid t = \sum_{i=0}^{d} \binom{n+i-1}{i} \le p \,\}$$
2. $d \leftarrow d + 1$ is the degree of the monomial m
3. $P \leftarrow p - t$, $N \leftarrow n - 1$, $r = [0, ..., 0]$, $i \leftarrow 1$
4. Until N becomes 0 or d becomes 0 repeat
   (a) Compute
   $$\max\{\, k \mid t = \sum_{i=0}^{k} \binom{N + (d-i) - 1}{d-i} \le P \,\}$$
   (if $\binom{N+(d-0)-1}{d} > P$ then $k \leftarrow -1$)
   (b) $r_i \leftarrow (k + 1)$, $N \leftarrow N - 1$, $d \leftarrow d - r_i$, $i \leftarrow i + 1$, $P \leftarrow P - t$
5. if $N = 0$ then set $r_i \leftarrow r_i + d$

Under the **graded reverse lexicographic ordering** we have the following mappings. The encoding of the monomial m is the integer:

$$\sum_{i=1}^{n} \binom{i + d_1 + ... + d_i - 1}{i}$$

Given a non-negative integer p, the corresponding monomial in n variables with exponent vector r is computed as follows:

1. Compute
$$\max\{\, d \mid t = \sum_{i=0}^{d} \binom{n+i-1}{i} \leq p \,\}$$
2. $d \leftarrow d+1$ is the degree of the monomial m
3. $r_n \leftarrow d$, $P \leftarrow p-t$, $N \leftarrow n-1$, $i \leftarrow n$
4. Until N becomes 0 or d becomes 0 repeat
   (a) Compute
   $$\max\{\, D \mid t = \sum_{i=0}^{D} \binom{N+i-1}{i} \leq P \,\}$$
   (b) $r_i \leftarrow r_i - (D+1)$, $N \leftarrow N-1$, $d \leftarrow d - r_i$, $i \leftarrow i-1$, $r_i \leftarrow d$, $P \leftarrow P - t$
5. if $d > 0$ then set $r_i \leftarrow d$

Under the **lexicographic ordering**, the monomials in $[x, y, z]$ in increasing order are $1 < z < z^2 < z^3 < z^4 < ....$ Under this ordering there is an infinite sequence of monomials in z before ever reaching monomials in higher order variables. Thus the only way to encode the monomials under this ordering would be to impose a bound on the degree. However, the degree of the intermediate polynomials when computing a Gröbner basis may grow quite large, hence imposing a bound on the degree may cause problems.

2.1.1 Implementation of Encodings

In the above encodings, the mappings involve the computation of many binomial coefficients which is quite expensive. We have worked out an improvement on the implementation which cut the runtime by 50%. The Maple code for the graded lexicographic and graded reverse lexicographic mappings is shown in the appendix, see inttomon_grlex, montoint_grlex and inttomon_grevlex, montoint_grevlex, respectively. The conversions require a linear number of integer multiplications and exact divisions.

3 Buchberger's Algorithm Over $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}$

Given a list of polynomials, F, in $\mathbb{Z}[x_1, ..., x_n]$, the polynomials are reduced mod a batch of primes $[p_1, ..., p_b]$ and stored in the data structure as described in the previous section. A Gröbner basis is then computed using Buchberger's algorithm working over $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}$.

In our implementation three orderings have been implemented, $>_{lex}$, $>_{grlex}$ and $>_{grevlex}$, see [1] for definitions. The S-pairs are stored in a heap data structure using the sugar selection strategy and all of the improvements mentioned previously have been implemented.

The issue of bad primes is addressed as follows. On each division with non-zero remainder $r = \big[\, [\, [c_1, ..., c_b], m_1 \,], \, ... \,\big]$, we check the list of leading coefficients $[c_1, ..., c_b]$. If $c_i = 0$ for some i then we have lost the leading term of r under the prime p_i. Hence p_i, with very high probability, is a bad prime and we remove p_i from the batch. In this manner we detect bad primes as soon as is possible.

The basis is reduced in the final step to guarantee uniqueness, which is needed for the Chinese remaindering step. For each batch, the resulting reduced Gröbner basis $G = \{g_1, ..., g_t\}$ is checked to ensure that it agrees with the previous result $\overline{G} = \{\overline{g_1}, ..., \overline{g_s}\}$ (if there is one). They agree if $t = s$ and $\mathrm{LM}(g_i) = \mathrm{LM}(\overline{g_i})$ for $i = 1, ..., t$. If they do not agree, all previous results are thrown away and we start anew.

3.1 Reconstruction

The Gröbner basis over $\mathbb{Q}$ is reconstructed by first applying the Chinese remainder algorithm and then applying rational reconstruction.

In our implementation, Maple's Chinese remaindering procedure, chrem, is used. chrem is mapped onto the list of coefficients of each term of each polynomial in the reduced Gröbner basis over $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}$, the encoded monomials are mapped back to Maple monomials and then Maple polynomials are constructed by summing the reconstructed terms. This gives us the Gröbner basis $G \in \mathbb{Z}_{p_1 \cdot ... \cdot p_b}[x_1, ..., x_n]$.

If there were previous Gröbner bases computed then we have $\overline{G} \in \mathbb{Z}_{\overline{m}}[x_1, ..., x_n]$ where $\overline{m}$ is the product of the primes from all previous batches. The implementation of Buchberger's algorithm returns a reduced Gröbner basis which is unique so the polynomials in G and $\overline{G}$ will contain the same leading monomials, hence we can apply Chinese remaindering to G and $\overline{G}$ to reconstruct a Gröbner basis over $\mathbb{Z}_m$ where $m = p_1 \cdot ... \cdot p_b \cdot \overline{m}$.

After Chinese remaindering we apply rational reconstruction. In our implementation, Maple's rational reconstruction procedure, iratrecon, is used. It works as follows. Given an image u mod m and positive integer bounds N and D, iratrecon will compute integers n and d such that $\frac{n}{d} \equiv u \bmod m$ where $|n| \leq N$, $|d| \leq D$. The reconstruction is unique if the bounds, N and D are such that $2ND < m$.

The procedure iratrecon is mapped onto the coefficients of the polynomials in the basis G with modulus m, the product of primes from all batches so far, and bound $N = D = \lfloor \sqrt{\frac{m}{2}} \rfloor - 1$, which guarantees uniqueness. Rational reconstruction may fail if there is not enough information for the reconstruction, that is, if m is not large enough. In this case, we get a new batch of primes and continue.

3.2 Termination Condition

Once rational reconstruction succeeds, the number of digits of agreement are checked. If they are sufficient then the Gröbner basis is returned. If we have b, m digit primes, then we can reconstruct $\frac{b \cdot m}{2}$ digits in the denominator and in the numerator. In our implementation we take 25 digits of agreement to be sufficient. The number of digits of agreement are checked as follows.

Let G be the Gröbner basis over $\mathbb{Q}$. For every polynomial $g \in G$, $g \leftarrow g/\text{icontent}(g)$, where icontent($g$) = GCD(numerators of the coefficients of g) / LCM(denominators of the coefficients of g). This makes the polynomials primitive over the integers.

Let $c = \max_{g \in G}\{\|g\|_\infty\}$, the coefficient with the largest absolute value. Then if $2 \cdot \lceil \log_{10} c \rceil \leq \lceil \log_{10} m \rceil - 25$, where m is the product of all primes, then we return G, otherwise we get another batch of primes and continue.

4 Results

This method was implemented in Maple V Release 5 as described previously and tested on various problems, varying the batch size from 1 to 20 for prime sizes of $5, 10, 20$ and 40 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: for a given prime size, 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.

4.1 Trinks

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

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

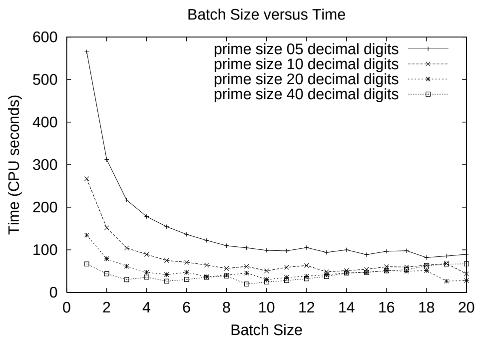


Figure 1: Trinks, $>_{lex}$, $b > s > p > t > z > w$

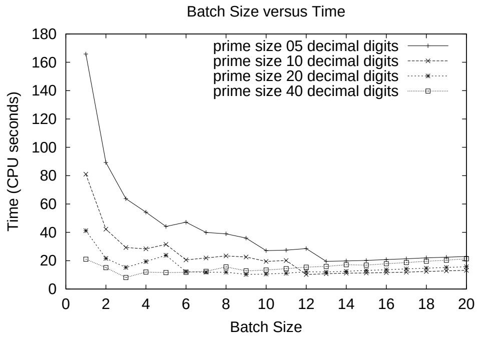


Figure 2: Trinks, $>_{grevlex}$, $b > s > p > t > z > w$

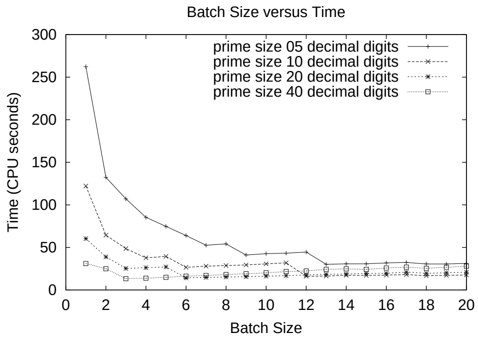


Figure 3: Trinks, $>_{grlex}$, $b > s > p > t > z > w$

4.2 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 $>_{lex}$, $>_{grevlex}$ and $>_{grlex}$ has 667, 62 and 61 decimal digits, respectively. The results of running the modular method on this problem under $>_{lex}$, $>_{grevlex}$ and $>_{grlex}$ are plotted in figures 4, 5, and 6, respectively.

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

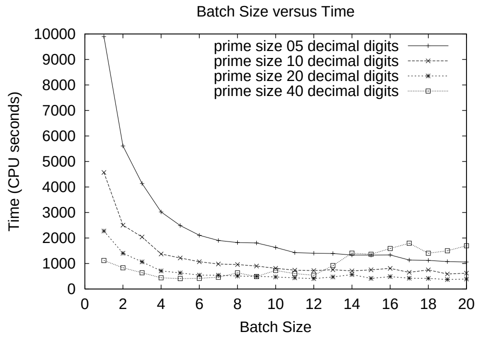


Figure 4: Cubics, $>_{lex}$, $x > y > z$

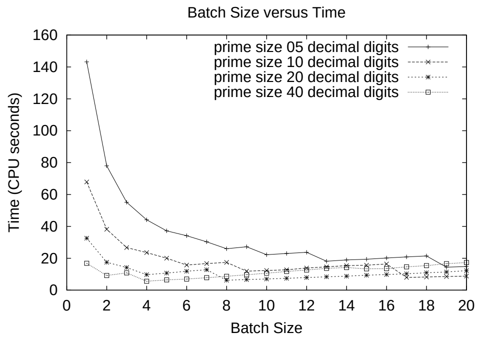


Figure 5: Cubics, $>_{grevlex}$, $x > y > z$

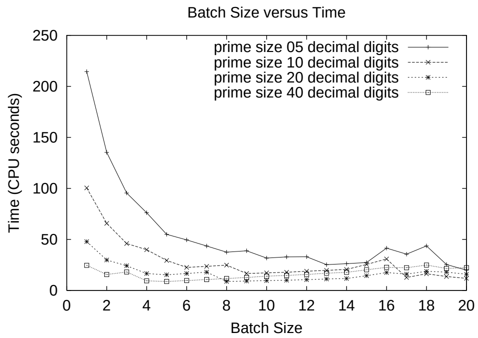


Figure 6: Cubics, $>_{grlex}$, $x > y > z$

4.3 The Katsura-4 Example

This system in $\mathbb{Q}[x, y, z, t, u]$ was taken from [3], where it was found to have bad coefficient blowup. The largest coefficient in the Gröbner basis of this system under $>_{lex}$, $>_{grevlex}$ and $>_{grlex}$ has 63, 22 and 23 decimal digits, respectively. The results of running the modular method on this problem under $>_{lex}$, $>_{grevlex}$ and $>_{grlex}$ are plotted in figures 7, 8 and 9, respectively.

$$\begin{aligned} \mathrm{K4} = [&2x^2 + 2y^2 + 2z^2 + 2t^2 + u^2 - u, \\ &xy + 2yz + 2zt + 2tu - t, \\ &2xz + 2yt + t^2 + 2zu - z, \\ &2xt + 2zt + 2yu - y, \\ &2x + 2y + 2z + 2t + u - 1] \end{aligned}$$

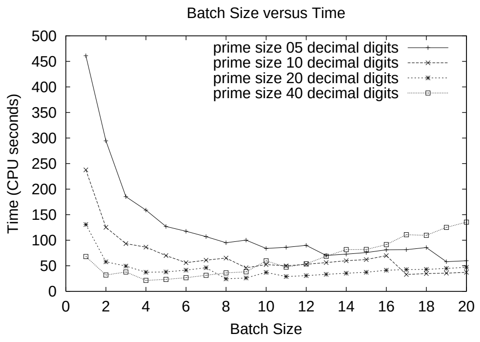


Figure 7: K4, $>_{lex}$, $x > y > z > t > u$

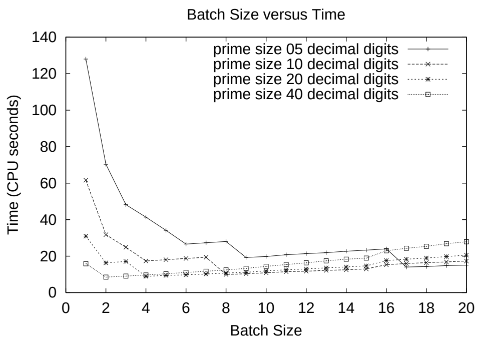


Figure 8: K4, $>_{grevlex}$, $x > y > z > t > u$

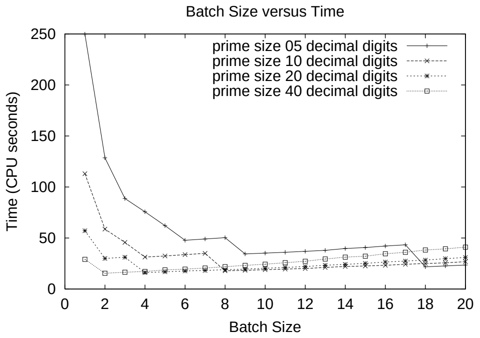


Figure 9: K4, $>_{grlex}$, $x > y > z > t > u$

5 Conclusion and Future Work

We sought to improve the efficiency of the modular method. The bottleneck is the monomial overhead. One improvement is to encode the monomials as integers. The main improvement is to compute the Gröbner basis mod primes $p_1, ..., p_b$ simultaneously.

This method performed better on these test problems under the lexicographic ordering than Maple's gbasis procedure. We ran the problems using the gbasis procedure in Maple V Release 4, which has a better implementation under the lexicographic ordering than Release 5, 6 or 7.

On the two hardest problems, Trinks and Cubics, Maple's gbasis failed to terminate in a reasonable amount of time (ie. less than one hour). The modular method performed very well on Trinks, achieving a worst time of 9 minutes (prime size 5, batch size 1) and a best time of 20 seconds (prime size 40, batch size 9). On Cubics the results varied much more widely. The modular method still achieved a reasonable best time of 6 minutes (prime size 20, batch size 19) but finished in a worst time of 2 hours and 45 minutes (prime size 5, batch size 1). However, Maple did not terminate at all. For the Katsura-4 example, Maple's gbasis finished in 20 minutes and the best time achieved by the modular method was 21 seconds (prime size 40, batch size 4), the worst was 8 minutes (prime size 5, batch size 1).

From these results we see that varying the prime size and the batch size can greatly affect the performance of this algorithm. We also see that in general the best times are achieved with larger prime sizes. For all of the test problems, the best times were achieved with prime sizes of 20 or 40. As the prime size increases, generally the batch size decreases. On the easier problems, batch sizes of 3 to 8 achieved the best times. On the harder problems batch sizes of 5 to 19 performed the best.

We have done a prototype implementation in Maple. The results indicate great potential in this method. However, we would expect to get more accurate timing results from an implementation in the C language. This would involve a lot of work in the form of memory management.

In the future we would like to implement the critical parts in C. The bottleneck in the current Maple implementation is the monomial arithmetic and vector operations in $\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_b}$. We could code these operations in C and then link the C subroutines to Maple. We are also considering an alternative representation for the polynomials in which we store the coefficients in a matrix and the monomials in a vector.

A Code for Monomial Encodings

```
#--------------------------------------------
# monomial encodings for grlex ordering

inttomon_grlex := proc(n::integer)
   proc(p::integer)
      local d,b,v,N,P,i,r;
      if n = 1 then RETURN([p]) fi;
      d := 0; b := 1; v := p - b;
      while v >= 0 do
         d := d + 1;
         b := iquo((n + d - 1)*b, d);
         v := v - b;
      od;
      P := v + b; N := n - 1;
      i := 1; r := [0$n];
      b := iquo(mul(N+d-j,j=1..N-1),(N-1)!);
      while d > 0 and N > 0 do
         v := P - b;
         while v >= 0 do
            r[i] := r[i] + 1;
            b := iquo(b*d, N + d - 1);
            d := d - 1;
            v := v - b;
         od;
         i := i + 1; N := N - 1;
         P := v + b;
         if N = 0 then r[i] := r[i] + d
         else b := iquo(b*N, N + d) fi;
      od;
      r;
   end
end:

montoint_grlex := proc()
   proc(x::list(integer))
      local n,d,i,j,k,b,p;
      n := nops(x);
      if n = 1 then RETURN(x[1]) fi;
      d := add(x[i], i = 1..n);
      p := iquo(mul(n+d-j,j=0..n-1),n!);
      b := p; k := 1;
      b := iquo(b*d, n + d);
      for i from n - 1 to 1 by -1 do
         b := iquo(b*(i + 1), i + d);
         for j to x[k] do
            b := iquo(b*(d - j), i + d - j)
         od;
         p := p - b;
         d := d - x[k];
         if d = 0 then break fi;
         k := k + 1;
      od;
      p - 1;
   end:
end:

#--------------------------------------------
# monomial encodings for grevlex ordering

inttomon_grevlex := proc(n::integer)
   proc(p::integer)
      local b,d,v,N,D,P,i,r;
      if n = 1 then RETURN([p]) fi;
```

```
      d := 0; b := 1;
      v := p - b;
      while v >= 0 do
         d := d + 1;
         b := iquo((n + d - 1)*b, d);
         v := v - b;
      od;
      P := v + b; N := n - 1;
      i := n; r := [0$n];
      while N > 0 and d > 0 do
         r[i] := d; b := 1;
         v := P - b; D := 0;
         while v >= 0 do
            D := D + 1;
            r[i] := r[i] - 1;
            b := iquo((N + D - 1)*b, D);
            v := v - b;
         od;
         N := N - 1; d := d - r[i];
         i := i - 1; P := v + b;
      od;
      if d > 0 then r[i] := d fi;
      r;
   end:
end:

montoint_grevlex := proc()
   proc(x::list(integer))
      local n,d,i,b,t,k;
      n := nops(x);
      if n = 1 then RETURN(x[1]) fi;
      d := add(x[i], i = 1..n);
      b := iquo(mul(n+d-j,j=1..n),n!);
      t := b;
      for k from n to 2 by -1
         while d - x[k] > 0 do
            t := iquo(t*k,k+d-1);
            for i to x[k] do
               t := iquo(t*(d-i),k+d-1-i);
            od;
            b := b + t; d := d - x[k];
         od;
      b;
   end:
end:
```

References

[1] 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.

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

[3] Alessandro Giovani, 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.

[4] D. Bayer and M. Stillman. *Macaulay Classic:* A computer algebra system for algebraic geometry, 1993. Available via anonymous ftp from ftp://math.harvard.edu/Macaulay.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.