On the Design and Implementation of Brown's Algorithm over the Integers and Number Fields

Michael B. Monagan
CECM, Simon Fraser University
Burnaby, British Columbia, V5A 1S6 Canada.
monagan@cecm.sfu.ca

Allan D. Wittkopf
CECM, Simon Fraser University
Burnaby, British Columbia, V5A 1S6 Canada.
wittkopf@cecm.sfu.ca

ABSTRACT

We study the design and implementation of the dense modular GCD algorithm of Brown applied to bivariate polynomial GCDs over the integers and number fields. We present an improved design of Brown's algorithm and compare it asymptotically with Brown's original algorithm, with GCDHEU, the heuristic GCD algorithm, and with the EEZGCD algorithm. We also make an empirical comparison based on Maple implementations of the algorithms. Our findings show that a careful implementation of our improved version of Brown's algorithm is much better than the other algorithms in theory and in practice.

1. INTRODUCTION

We investigate the problem of computing greatest common divisors of bivariate polynomials over the integers $\mathbb{Z}$ and a number field $\mathbb{Q}(\alpha)$. Our interest in bivariate GCDs resulted from GCD problems given to us by Edgardo Cheb-Terrab (in the context of finding invariants of Abel ODE, see [4]) and David Boyd (in the context of computing A-polynomials). These are large, rather dense, bivariate GCD problems, created by an elimination procedure, for which Brown's algorithm [2] should be a good choice. In section 2 we outline his algorithm and show that it is inefficient, however, when the GCD is small. In section 3 we detail an improved version of Brown's algorithm which is always efficient when the GCD is small. In section 4 we compare asymptotically Brown's algorithm and our improved version of it with GCDHEU, the heuristic GCD algorithm of Char et al. [3], and the EEZGCD algorithm of Wang [11]. We find that our improved version is asymptotically better on two important classes of GCD problems. In section 5 we compare Maple implementations of the three algorithms and find that our improved version of Brown's algorithm is much faster than the GCDHEU and EEZGCD implementations. Our contention is that most bivariate GCD problems in practice are not sparse enough to warrant the use of a sparse GCD algorithm like the EEZGCD algorithm and since our modified version of Brown's algorithm is so much faster on even semi-sparse problems, it is the algorithm of choice. In section 6 we use our improved version of Brown's algorithm to compute bivariate GCDs over number fields. We find that in order to obtain good performance when a number field has low degree, we must map the computation into $\mathbb{Z}_p$. Throughout the paper we will use the following notations:

$||a||_\infty$ denotes the height of a polynomial, that is, the magnitude of the largest integer coefficient.

$\mathrm{lc}_{x,y} : R[x,y] \to R$ is the leading coefficient of a polynomial using lexicographical order with $x > y$.

$\mathrm{cont}_{x,y} : R[x,y] \to R$ is the content of a polynomial, namely the GCD of the coefficients in R.

$\mathrm{pp}_{x,y} : R[x,y] \to R[x,y]$ is the primitive-part of a polynomial, namely the polynomial divided by it's content.

$\Delta_{x,y} : R[x,y] \to \mathbb{N}^2$ is the vector degree of a polynomial using lexicographical order with $x > y$.

2. BROWN'S GCD ALGORITHM

Brown's description of the "dense modular GCD algorithm" in [2] is reproduced here for $\mathbb{Z}[x,y]$ in sufficient detail for the purpose of asymptotic analysis. Brown's algorithm has two subroutines, M and P. The main subroutine M maps a polynomial GCD computation from $\mathbb{Z}[x,y]$ into one or more GCD computations in $\mathbb{Z}_p[x,y]$ for $p \in p_0, p_1, \ldots$ and applies the Chinese remainder theorem to the resulting solutions. Brown's presentation of subroutine P is similar. It maps a polynomial GCD computation from $\mathbb{Z}_p[y][x]$ into one or more GCD computations in $\mathbb{Z}_p[x]$ by reducing the input polynomials modulo $y - \alpha$ for sufficiently many $\alpha \in \mathbb{Z}_p$ and recovering y, again, by application of the Chinese remainder theorem.

An important observation to make about Brown's algorithm is that both subroutines have been deliberately designed so that their termination does not require a trial division of the inputs by the GCD, nor multiplication of the GCD and its cofactors to verify that the results are correct. This feature leads to good performance on some GCD problems but poor performance on others.

[*]This material is based on work supported in part by NSERC of Canada.
Version 5/17/2000 (4:4).

Algorithm M (Brown 1971)

Input: $a, b \in \mathbb{Z}[x,y] \setminus \{0\}$
Output: $g = \mathrm{GCD}(a,b)$, $\bar{a} = a/g$, $\bar{b} = b/g$

```
 1 Compute integer contents and their GCD.
   $ca = \mathrm{cont}_{x,y}(a) \in \mathbb{Z}$, $a = a/ca$, $cb = \mathrm{cont}_{x,y}(b) \in \mathbb{Z}$, $b = b/ca$, and $cg = \mathrm{GCD}(ca, cb)$.
 2 Compute the correction coefficients.
   Set $la = \mathrm{lc}_{x,y}(a) \in \mathbb{Z}$, $lb = \mathrm{lc}_{x,y}(b) \in \mathbb{Z}$, and $\gamma = \mathrm{GCD}(la, lb)$.
 3 Estimate the bound for twice the height of $\gamma \times g$, $\gamma \times \bar{a}$, and $\gamma \times \bar{b}$ as $B = 2\gamma \max(||a||_\infty, ||b||_\infty)$.
 4 Set $m = 1$ the product of the moduli.
   Set $\mathbf{e} = \min(\Delta_{x,y} a, \Delta_{x,y} b)$.
Loop:
 6 Choose a new prime $p$ that does not divide $la$ or $lb$.
 7 Compute $a_p = a \bmod p, b_p = b \bmod p$.
 8 Compute $g_p, \bar{a}_p, \bar{b}_p \in \mathbb{Z}_p[x,y]$ the monic GCD of $a_p$ and $b_p$ and their cofactors using algorithm M. If M fails (too few evaluation points are available) goto Loop.
 9 Test for $a, b$ relatively prime.
   If $\Delta_{x,y} g_p = \mathbf{0}$ then output $cg$, $(ca/cg)a$, $(cb/cg)b$.
10 If $\Delta_{x,y} g_p > \mathbf{e}$ then (skip this prime) goto Loop.
11 Leading coefficient correction in $\mathbb{Z}$.
   Set $g_p = \gamma \times g_p \bmod p$, $\bar{a}_p = \gamma^{-1} a_p \bmod p$, $\bar{b}_p = \gamma^{-1} b_p \bmod p$.
12 First image?
   If $m = 1$ then set $g_m = g_p$, $\bar{a}_m = \bar{a}_p$, $\bar{b}_m = \bar{b}_p$, $m = p$, $\mathbf{e} = \Delta_{x,y} g_p$ and goto Loop.
13 If $\Delta_{x,y} g_p < \mathbf{e}$ then all previous primes were unlucky. Restart the algorithm keeping only the current prime.
   Set $g_m = g_p$, $\bar{a}_m = \bar{a}_p$, $\bar{b}_m = \bar{b}_p$, $m = p$, $\mathbf{e} = \Delta_{x,y} g_p$ and goto Loop.
14 Combine the new image with the old using the CRA and express the result in the symmetric range for $\mathbb{Z}_p$.
   Set $g_m = \mathrm{CRA}(\ [g_p, g_m], [p, m]\ )$,
   Set $\bar{a}_m = \mathrm{CRA}(\ [\bar{a}_p, \bar{a}_m], [p, m]\ )$,
   Set $\bar{b}_m = \mathrm{CRA}(\ [\bar{b}_p, \bar{b}_m], [p, m]\ )$.
15 Set $m = m \times p$, if $m \le B$ then goto Loop.
16 Test for termination.
   Without explicitly computing $g_m \times \bar{a}_m$ and $g_m \times \bar{b}_m$ determine if $||g_m \times \bar{a}_m||_\infty < m/2$ and $||g_m \times \bar{b}_m||_\infty < m/2$. If so then we have $m | a - g_m \times \bar{a}_m$ and $m | b - g_m \times \bar{b}_m$ and also $a - g_m \times \bar{a}_m = 0$ and $b - g_m \times \bar{b}_m = 0$ over $\mathbb{Z}$ hence we are done; so
   set $g = \mathrm{pp}(g_m))$, $\delta = \mathrm{lc}_{x,y}\, g \in \mathbb{Z}$, $\bar{a} = \bar{a}_m/\delta$, $\bar{b} = \bar{b}_m/\delta$.
   Output $cg \times g$, $(ca/cg)\bar{a}$, $(cb/cg)\bar{b}$.
17 Goto Loop – either we do not have enough primes yet or all primes are unlucky.
```

Algorithm P (Brown, 1971)

Input: $a, b \in \mathbb{Z}_p[y][x] \setminus \{0\}$, p prime.
Output: $g = \mathrm{GCD}(a,b)$, $\bar{a} = a/g$, $\bar{b} = b/g$

```
 1 Compute contents in $\mathbb{Z}_p[y]$ and their GCD.
   Set $ca = \mathrm{cont}_x a \in \mathbb{Z}_p[y]$, $a = a/ca$, $cb = \mathrm{cont}_x b \in \mathbb{Z}_p[y]$, $b = b/ca$, and $cg = \mathrm{GCD}(ca, cb)$.
 2 Compute the correction coefficients.
   Set $la = \mathrm{lc}(a) \in \mathbb{Z}_p[y]$, $lb = \mathrm{lc}(a) \in \mathbb{Z}_p[y]$, $\gamma = \mathrm{GCD}(la, lb)$.
 3 Bound the degree in $y$ of $\gamma \times g$, $\gamma \times \bar{a}$, $\gamma \times \bar{b}$.
   Set $B = \deg_y \gamma + max(\deg_y a, \deg_y b)$.
 4 Set $m = 1$ the product of the moduli.
   Set $e = \min(\deg_y a, \deg_y b) \in \mathbb{Z}$.
Loop:
 6 Choose a new evaluation point $\alpha \in \mathbb{Z}_p$ such that $y - \alpha$ does not divide $la \times lb$. If no such evaluation point exists then $p$ is too small and the algorithm fails.
 7 Compute $a_\alpha = a \bmod (y - \alpha) \in \mathbb{Z}_p[x]$, and $b_\alpha = b \bmod (y - \alpha) \in \mathbb{Z}_p[x]$.
 8 Compute $g_\alpha, \bar{a}_\alpha, \bar{b}_\alpha \in \mathbb{Z}_p[x]$ the monic GCD of $a_\alpha$ and $b_\alpha$ and their cofactors using the Euclidean algorithm.
 9 Test for $a, b$ relatively prime.
   If $\deg_x g_\alpha = 0$ then output $cg$,$(ca/cg)a$,$(cb/cg)b$.
10 If $\deg_x g_\alpha > e$ then (skip this prime) goto Loop.
11 Leading coefficient correction in $\mathbb{Z}_p[y]$.
   Set $g_\alpha = \gamma(\alpha) \times g_\alpha \bmod p$, $\bar{a}_\alpha = \gamma(\alpha)^{-1} \bar{a}_\alpha \bmod p$, and $\bar{b}_\alpha = \gamma(\alpha_n)^{-1} \bar{b}_\alpha \bmod p$.
12 First image?
   If $m = 1$ then set $g_m = g_\alpha$, $\bar{a}_m = \bar{a}_\alpha$, $\bar{b}_m = \bar{b}_\alpha$, $m = p$, $e = \deg_x g_\alpha$ and goto Loop.
13 If $\deg_x g_\alpha < e$ then all previous points were unlucky. Restart the algorithm keeping only the current point.
   Set $g_m = g_\alpha$, $\bar{a}_m = \bar{a}_\alpha$, $\bar{b}_m = \bar{b}_\alpha$, $m = y - \alpha$, $e = \deg_x g_\alpha$ and goto Loop.
14 Combine the new image with the old using the CRA.
   Set $g_m = \mathrm{CRA}(\ [g_\alpha, g_m], [y - \alpha, m]\ )$,
   Set $\bar{a}_m = \mathrm{CRA}(\ [\bar{a}_\alpha, \bar{a}_m], [y - \alpha, m]\ )$,
   Set $\bar{b}_m = \mathrm{CRA}(\ [\bar{b}_\alpha, \bar{b}_m], [y - \alpha, m]\ )$.
15 Set $m = m \times (y - \alpha)$.
   If $\deg_y m \le B$ then goto Loop.
16 Test for termination.
   If $\deg_y g_m + \deg_y \bar{a}_m = \deg_y \gamma + \deg_y a$ and $\deg_y g_m + \deg_y \bar{b}_m = \deg_y \gamma + \deg_y b$ then set $g = \mathrm{pp}_x(g_m)$, $\delta = \mathrm{lc}_x\, g$, $\bar{a} = \bar{a}_m/\delta$, $\bar{b} = \bar{b}_m/\delta$, and output $cg \times g$, $(ca/cg)\bar{a}$, $(cb/cg)\bar{b}$.
17 Goto Loop – all $\alpha's$ are unlucky.
```

In order to determine the complexity of Brown's algorithm, the central quantities of interest will be the number of primes p used by subroutine M and the number of evaluation points α used by subroutine P. The reader will note that these quantities, computed in step [3] and used in step [15] of both subroutines, are determined by the size of the inputs a and b and not by the size of $g, \bar{a}$ and $\bar{b}$. In order to improve the asymptotic time complexity of Brown's algorithm we first identify two classes of GCD problems which are common in practice.

Balanced GCD Problems

Suppose $||\bar{a}||_\infty, ||\bar{b}||_\infty$ and $||g||_\infty < 10^n$. Suppose also that $\deg_x \bar{a} = \deg_x \bar{b} = \deg_x g = n$ and $\deg_y \bar{a} = \deg_y \bar{b} = \deg_y g = n$. Thus in this class of GCD problems we assume that the size of the coefficients grows proportionately with the degree of the polynomials in both variables. If the polynomials $g, \bar{a}$ and $\bar{b}$ are also dense then their size is $O(n^3)$ and the cost of computing $\bar{a} \times g$ or a/g using classical algorithms for polynomial multiplication and division is $O(n^6)$. Suppose instead that the polynomials $g, \bar{a}$ and $\bar{b}$ have $O(n)$ terms instead of $O(n^2)$. We will call such polynomials semi-sparse. Their size will be $O(n^2)$ and the cost of the classical multiplication and division algorithms reduce to $O(n^4)$. As the polynomials become completely sparse, for example, of the form $c_0 + c_1 x^n y^n$, the classical multiplication and division algorithms reduce to $O(n^2)$, the cost of multiplying the n digit coefficients.

Small GCD Problems

Suppose $||\bar{a}||_\infty < 10^n$ and $||\bar{b}||_\infty < 10^n$ and $\deg_x \bar{a} = \deg_y \bar{a} = n$ and $\deg_x \bar{b} = \deg_y \bar{b} = n$. Suppose $||g||_\infty = O(1)$ and $\deg_x g = \deg_y g = O(1)$. Thus for this problem we assume that for the cofactors, the size of the coefficients grows proportionately with the degree in both variables but the size of the GCD remains fixed. We are thinking of the case when the size of the GCD is small e.g. degree 2 in x and y but the size of the cofactors is large, e.g. degree 18 in x and y. If $\bar{a}$ and $\bar{b}$ are dense then the cost of computing $\bar{a} \times g$ using the classical algorithms for polynomial multiplication and long division is $O(n^3)$. If the polynomials $\bar{a}$ and $\bar{b}$ are semi-sparse, each having $O(n)$ terms, then their size is $O(n^2)$. The cost of multiplication and division is $O(n^2)$.

The cost of Brown's algorithm can be shown to be $O(n^4)$ for all non-trivial GCD problems (see section 4). Although Brown's algorithm is very good for the balanced GCD problem, it is very poor on the small GCD problem, as the $O(n)$ and $O(n^2)$ factor of difference is significant. In the next section we redesign the algorithm so that it is always fast for small GCD problems.

3. MODIFIED BROWN'S ALGORITHM

The primary source of inefficiency in Brown's algorithm occurs when the number of modular images computed by algorithms M or P is much more than necessary to reconstruct g. In subroutine M, if $\Delta g \neq \mathbf{0}$, the number of images required must be enough to reconstruct $\gamma \times g$, $\gamma \times \bar{a}$ and $\gamma \times \bar{b}$. Brown uses estimates for the size of these quantities based on $||a||_\infty$ and $||b||_\infty$ in step [3]. It is well known that instead of lifting all of $g, \bar{a}, \bar{b}$ and stopping when $a - g\bar{a} = 0$ and $b - g\bar{b} = 0$, one may instead lift g one image at a time stopping if $g|a$ and $g|b$. By performing the trial divisions $g|a$ and $g|b$ only when g does not change from one image to the next, we will, with high probability, make these trial divisions once. This means that the algorithm will almost always use one more prime than necessary to reconstruct $(\gamma/\delta)\ g$.

The same problem occurs in subroutine P and one could employ the same idea to resolve it, but instead in our improved version of Brown's algorithm (algorithm G below) we propose the following. We precompute (tight) upper degree bounds dg_x and dg_y for, respectively, $\deg_x g$ and $\deg_y g$. This will be done by algorithm B below by computing a single univariate GCD in $\mathbb{Z}_p[x]$ and $\mathbb{Z}_p[y]$ respectively. If x is chosen to be the main variable, we will use dg_y to determine how many evaluation points are needed, and interpolate all the images in a single step rather than incrementally, as this will avoid creating a lot of intermediate polynomials. Further, we do not do the trial divisions for the interpolated result (see statement 19 in algorithm G), as the trial divisions (see statement 27) are sufficient to verify that the final result is correct.

At the same time that we compute dg_x and dg_y we also compute degree bounds on the contents in each variable. This enables us to determine if $g = 1$ (see statement 2), to reduce the number of evaluation points needed for y when the content is not 1 (see statement 11) and to detect another source of unlucky primes (see statement 9). Further, we use these degree bounds dg_x, dg_y, and dc_y to detect an unlucky prime or evaluation point as early as is possible. Thus we present the algorithm as a single subroutine so that it can immediately restart itself with a new prime instead of completing a modular GCD computation only to discard it. This makes for a less transparent presentation but it does mean that the presence of unlucky primes or evaluation points have a negligible affect on the algorithm.

Algorithm B

Inputs: $a, b \in \mathbb{Z}[x,y] \setminus \{0\}$.
Output: (dg, dc) satisfying $dg \geq \deg_x \mathrm{GCD}(a,b)$ and $dc \geq \deg_y GCD(\mathrm{cont}_x(a), \mathrm{cont}_x(b))$

```
1 Compute $\gamma \in \mathbb{Z} = \mathrm{GCD}(\mathrm{lc}_{x,y}(a), \mathrm{lc}_{x,y}(b))$
2 Choose a prime $p$ that does not divide $\gamma$.
3 Set $a_p \in \mathbb{Z}_p[x,y] = a \bmod p$, $b_p \in \mathbb{Z}_p[x,y] = b \bmod p$.
4 Compute and the contents and their GCD.
  Set $ca \in \mathbb{Z}_p[y] = \mathrm{cont}_x(a_p)$, $cb \in \mathbb{Z}_p[y] = \mathrm{cont}_x(b_p)$,
  and $cg = \mathrm{GCD}(ca, cb)$. Set $a_p = a_p/ca,\ b_p = b_p/cb$.
5 Compute $\gamma \in \mathbb{Z}_p[y] = \mathrm{GCD}(\mathrm{lc}_x(a_p), \mathrm{lc}_x(p_b))$.
6 Choose $\alpha \in \mathbb{Z}_p$ such that $\gamma(\alpha) \not\equiv 0 \bmod p$. If no such
  $\alpha$ exists then output $\min(\deg_x a_p, \deg_x b_p)$, $\deg_y c_p$.
7 Compute $g_{p\alpha} \in \mathbb{Z}_p[x] = \mathrm{GCD}(a_p(x, y{=}\alpha), b_p(x, y{=}\alpha))$.
8 Output $\deg_x g_{p\alpha}$, $\deg_y cg$.
```

Algorithm G

Inputs: $a, b \in \mathbb{Z}[x,y] \setminus \{0\}$
Output: $g = \mathrm{GCD}(a,b)$, $\bar{a} = a/g$, $\bar{b} = b/g$

```
1 Compute integer contents and their GCD.
  Set $ca = \mathrm{cont}_{x,y}(a) \in \mathbb{Z}$, $a = a/ca$, $cb = \mathrm{cont}_{x,y}(b) \in$
  $\mathbb{Z}$, $b = b/cb$, $cg = GCD(ca, cb)$.
2 Compute degree bounds $dg_x, dc_y$ satisfying $dg_x \geq$
  $\deg_x g$ and $dc_y \geq \deg_y \mathrm{GCD}(\mathrm{cont}_x(a), \mathrm{cont}_x(b))$ using
  algorithm B.
  If $dg_x = dc_y = 0$ then output $cg$, $(ca/cg)a$, $(cb/cg)b$.
  Similarly compute degree bounds $dg_y, dc_x$ satisfying
  $dg_y \geq \deg_y g$ and $dc_x \geq \deg_x GCD(\mathrm{cont}_y(a), \mathrm{cont}_y(b))$
  using algorithm B.
  If $dg_y = dc_x = 0$ then output $cg$, $(ca/cg)a$, $(cb/cg)b$.
3 Choose the main variable $x$ and the minor variable $y$.
4 Compute the correction coefficient for $\mathbb{Z}$.
  Set $\gamma = \mathrm{GCD}(\ \mathrm{lc}_{x,y}(a), \mathrm{lc}_{x,y}(b)\ )$
5 To ensure that the degree of the GCD computed mod
  $p$ will not be less than the degree of $g$ in both $x$ and
  $y$, $p$ must not divide the integer badprimes.
  Set badprimes $= \gamma\ \mathrm{GCD}(\ \mathrm{lc}_{y,x}(a), \mathrm{lc}_{y,x}(b)\ )$
```

```
    6 Set $m = 1$ the product of the moduli.
Nextprime
      Choose a new prime $p$ that does not divide $badprimes$.
    7 Compute $a_p = a \bmod p,\ b_p = b \bmod p$.
    8 Compute the contents in $\mathbb{Z}_p[y]$ and their GCD.
      Set $ca_p = \mathrm{cont}_x(a)$, $a_p = a_p/ca_p$, $cb_p = \mathrm{cont}_x(b)$, $b_p = b_p/cb_p$, $cg_p = GCD(ca_p, cb_p)$.
    9 Detect unlucky primes.
      If $\deg_y cg_p > dc_y$ then goto Nextprime.
      If $\deg_y cg_p < dc_y$ then (bound and all previous primes were unlucky) set $dc_y = \deg_y cg_p$, $m = 1$ and continue.
   10 Compute the correction coefficient in $\mathbb{Z}_p[y]$.
      Set $\gamma_y = \mathrm{GCD}(\mathrm{lc}_x(a_p), \mathrm{lc}_x(b_p))$.
   11 Bound the number of evaluation points needed less 1.
      Set $N = \min(\deg_y a - \deg_y ca_p, \deg_y b - \deg_y cb_p, dg_y - dc_y + \deg_y \gamma_y)$.
   12 Set $n = 0$.
Nexteval
        13 Choose a new evaluation point $\alpha_n \in \mathbb{Z}_p$ such that $\gamma_x(\alpha_n) \not\equiv 0 \bmod p$. If no such evaluation point exists then go to Nextprime.
        14 Evaluate at $y = \alpha_n$. Set $a_{pn} = a_p(x, y{=}\alpha_n) \bmod p$, $b_{pn} = b_p(x, y{=}\alpha_n) \bmod p$.
        15 Compute $g_{pn} \in \mathbb{Z}_p[x]$ the monic GCD of $a_{pn}$ and $b_{pn}$ using the Euclidean algorithm.
        16 Detect unlucky $\alpha_n$
           If $\deg_x(g_{pn}) > dg_x$ then goto Nexteval.
           If $\deg_x g_{pn} < dg_x$ then set $m = 1$, $dg_x = \deg_x g_{pn}$ and goto Nextprime.
        17 Leading coefficient correction.
           Set $g_{pn} = \gamma_y(\alpha_n) \times g_{pn} \bmod p$.
        18 Set $n = n + 1$. If $n \leq N$ then goto Nexteval.
   19 Interpolate $y$.
      $g_p = \mathrm{Interpolate}([\alpha_0, \ldots, \alpha_N], [g_{p0}, \ldots, g_{pN}])$.
   20 Divide out spurious content in $\mathbb{Z}_p[y]$, and multiply by true content. $g_p = cg_p \times \mathrm{pp}_x(g_p)$.
   21 Check for unlucky primes.
      If $\deg_y g_p > dg_y$ then goto Nextprime.
      If $\deg_y g_p < dg_y$ then set $m = 1$, $dg_y = \deg_y g_p$ and goto Nextprime;
   22 Leading coefficient correction in $\mathbb{Z}$. $g_p = \gamma \times g_p \bmod p$
   23 Put $g_p$ in the symmetric range mod $p$.
   24 First image?
      If $m = 1$ then set $m = p$, $h = g_p$ and goto Nextprime;
   25 Combine the new image with the old using the CRA.
      Set $g_m = \mathrm{CRA}([g_p, h], [p, m])$, $m = p \times m$.
   26 If $g_m \neq h$ then set $h = g_m$ and goto Nextprime.
   27 We very probably have enough primes.
      Set $g = \mathrm{pp}_{x,y}(g_m)$.
      If $g|a$ and $g|b$ then output $cg \times g$, $a/g$, $b/g$.
   28 Goto Loop - either we do not have enough primes yet or all primes were unlucky.
```

4. ASYMPTOTIC COMPARISON

We investigate the asymptotic complexity of the four algorithms, Brown's algorithm, our improved version of Brown's algorithm, the heuristic algorithm GCDHEU of Char et al. [3], and the EEZGCD algorithm of Wang [11]. We consider two classes of GCD problems, the balanced GCD problems and small GCD problems defined in section 2, and for each class, dense and semi-sparse polynomials. These problems occur frequently in practice; the small GCD problem perhaps being the most frequent kind of GCD problem. For both problems let $g = GCD(a, b)$, $a, b, g \in Z[y, x]$ and let $\bar{a} = a/g$ and $\bar{b} = b/g$ be the cofactors.

The Balanced GCD Problem

For the dense balanced GCD problem we have already stated the complexity of computing $g \times \bar{a}$ using classical multiplication is $\mathrm{O}(n^6)$, computing a/g using classical division is $\mathrm{O}(n^6)$, and Brown's algorithm is $\mathrm{O}(n^4)$. The results for the GCD algorithms assume that no unlucky primes or evaluations occur and we have dropped any $\log\log$ factors from the complexity estimates.

Result 1: The asymptotic complexity of Brown's algorithm for computing g the GCD of a and b is $\mathrm{O}(n^4)$.

Sketch of proof: Brown's algorithm uses $\mathrm{O}(n)$ primes since $2||a||_\infty \leq B \leq 2\,\mathrm{lc}(a)||a||_\infty$ and these quantities are both $\mathrm{O}(10^n)$. Each modular reduction is $\mathrm{O}(n^3)$ because there are $\mathrm{O}(n^2)$ coefficients to reduce and each is $\mathrm{O}(n)$ digits long. Each modular GCD is $\mathrm{O}(n^3)$ because $\mathrm{O}(n)$ evaluation points are made (each evaluation costs $\mathrm{O}(n^2)$, each univariate GCD is $\mathrm{O}(n^2)$, and the Chinese remaindering cost for each coefficient in y is $\mathrm{O}(n^2)$). Thus the $\mathrm{O}(n)$ modular reductions in subroutine P are $\mathrm{O}(n^4)$ and the $\mathrm{O}(n)$ modular GCDs in subroutine M are also $\mathrm{O}(n^4)$. The Chinese remaindering in subroutine M is also $\mathrm{O}(n^4)$ because there are $\mathrm{O}(n^2)$ coefficients in g to be reconstructed, each of which is $\mathrm{O}(n)$ digits in length.

Result 2: The asymptotic complexity of our improved version of Brown's algorithm is $O(n^4) + D(n)$ where $D(n)$ is the cost of dividing a/g and b/g.

The $\mathrm{O}(n^4)$ term is essentially the same as Brown's algorithm with the main difference being that polynomial interpolation replaces Chinese remaindering (though they are equivalent). Note the cost of the trial divisions $D(n)$ is $\mathrm{O}(n^6)$ if classical integer and polynomial arithmetic is used but this can be reduced to $\mathrm{O}(n^4)$ if a modular algorithm is also used for division in $\mathbb{Z}[x, y]$.

Result 3: The asymptotic complexity of algorithm GCDHEU is $O(n^6) + D(n)$ when it succeeds where $D(n)$ is the cost of dividing a/g and b/g.

Sketch of proof: In order to recover the variable y the evaluation point α must bound $2||g||_\infty$ which is $\mathrm{O}(n)$ digits in length. The algorithm uses $\alpha = 2\min(||a||_\infty, ||b||_\infty)$ which is $\mathrm{O}(n)$ digits in length. In order to recover the x variable the evaluation point β must bound $||g(y = \alpha, x)||_\infty$ which will be $\mathrm{O}(n^2)$ digits in length. The algorithm uses $\beta = 2\min(||a(y = \alpha, x)||_\infty, ||b(y = \alpha, x)||_\infty)$ which is $\mathrm{O}(n^2)$ digits in length. The integer GCD computation of

$a(x=\beta, y=\alpha)$ and $b(x=\alpha, y=\beta)$ will therefore be of size $O(n^3)$ digits in length. Assuming that integer GCD is quadratic, this is $O((n^3)^2) = O(n^6)$ in total.

Result 4: The asymptotic complexity of algorithm EEZGCD will depend on how the integer coefficients are handled and whether the algorithm uses trial division to terminate or waits until $a - g\bar{a} = 0$ over $\mathbb{Z}$. The computation is normally done modulo p^l where p is a prime and $p^l > 2||a||_\infty$ where a is the lifting polynomial, though Chinese remaindering could be used here. In this case the modulus p^l used will have to be $O(n)$ digits long. The asymptotic complexity of the EEZGCD algorithm will be no better than $O(n^6) + D(n) + F(n)$ where $D(n)$ is the cost of dividing b/g and $F(n)$ is the cost of factoring the leading coefficient of the lifting polynomial a.

Sketch of proof: The number of arithmetic operations done in $\mathbb{Z}_{p^l}$ by the Hensel lifting algorithm is $O(n^4)$ – see Miola and Yun [9] or Bernardin [1] – hence the result $O(n^6)$.

Suppose that the polynomials $g, \bar{a}$ and $\bar{b}$ have $O(n)$ terms (they are semi-sparse) then their total size is $O(n^2)$. We state without proof the results in Table 1.

	Dense case	Semi-sparse case
Size of a, b	$O(n^3)$	$O(n^2)$
Multiplication	$O(n^6)$	$O(n^4)$
Division	$O(n^6)$	$O(n^4)$
Brown	$O(n^4)$	$O(n^4)$
Improved Brown	$O(n^4)$ + D(n)	$O(n^4)$ + D(n)
GCDHEU	$O(n^6)$ + D(n)	$O(n^6)$ + D(n)
EEZGCD	$O(n^6)$ + D(n)	$O(n^5)$ + D(n) + F(n)

Table 1: Complexity Results for Balanced GCD

These results indicate that for balanced GCD problems, the time complexity of the improved version of Brown's algorithm is better than GCDHEU and EEZGCD but that it might not be as good as Brown's original algorithm if the division algorithm is more expensive than $O(n^4)$.

The Small GCD Problem

We consider two cases. In the dense case $\bar{a}$ and $\bar{b}$ are dense of degree n in both variables with n digit coefficients. In the semi-sparse case $\bar{a}$ and $\bar{b}$ have $O(n)$ non-zero terms and are of degree n in both variables with n digit coefficients. In both cases the GCD g has degree $O(1)$ in both variables and coefficients of size $O(1)$ digits.

	Dense case	Semi-sparse case
Size of a, b	$O(n^3)$	$O(n^2)$
Multiplication	$O(n^3)$	$O(n^2)$
Division	$O(n^3)$	$O(n^2)$
Brown	$O(n^4)$	$O(n^4)$
Improved Brown	$O(n^3)$	$O(n^2)$
GCDHEU	$O(n^6)$	$O(n^6)$
EEZGCD	$O(n^4)$	$O(n^4)$

Table 2: Complexity Results for Small GCD

This is the case where our improved version of Brown's algorithm is clearly the best overall and, importantly, it has the same complexity as classical polynomial division and multiplication. It will in both cases, with high probability, use $O(1)$ primes and $O(1)$ evaluation points. The cost is dominated by the $O(n^3)$ modular reduction cost in the dense case since there are $O(n^2)$ coefficients to reduce, each of which is $O(n)$ digits long. In the semi-sparse case the modular reduction is $O(n^2)$ since there are $O(n)$ coefficients each of which is $O(n)$ digits long. The evaluation and univariate GCD cost is also $O(n^2)$.

5. IMPLEMENTATION

We have implemented our improved version of Brown's algorithm for $\mathbb{Z}[x,y]$ in Maple V Release 5. In the following we will refer to it as the DIVBRO algorithm. We make an empirical comparison of it with the implementation of the GCDHEU and EEZGCD algorithms in Maple for both the balanced GCD problem and small GCD problems. We first provide some details of the DIVBRO, GCDHEU and EEZGCD implementations in Maple.

The DIVBRO Implementation

We represent polynomials in $\mathbb{Z}_p[x,y]$ as Maple lists of $modp1$ polynomials. The modp1 data structure is described in [10]. It is a dense representation where a polynomial $a = \sum_{i=0}^n a_i y^i \in \mathbb{Z}_p[y]$ is represented by an array of machine integers $[a_0, a_1, \ldots, a_n]$. Thus the data structure for $\mathbb{Z}_p[y][x]$ is a recursive dense structure.

A good implementation of any GCD algorithm will consider which of the variables to choose as the main variable. Taking into account only the information about the degree of the input polynomials in the variables, a reasonable choice would be the following. Choose x to be the main variable if

$$\deg_x a \times \deg_x b \leq \deg_y a \times \deg_y b$$

and y otherwise, minimizing the expected cost of the individual univariate GCD computations. In our case, we have precomputed degree bounds dg_x on $\deg_x g$ and dg_y on $\deg_y g$. We use all four quantities to more accurately estimate the evaluation, univariate GCD, and interpolation costs of the algorithm with x chosen as the main variable to be

$$(dg_y + 1)(\deg_y a \deg_x a + \deg_y b \deg_x b),$$
$$(dg_y + 1)(\deg_x a - dg_x + 1)(\deg_x b - dg_x + 1),$$
$$\text{and } (dg_y + 1)dg_x(dg_y + 1),$$

respectively. We choose x to be the main variable if the sum of those three quantities is less than the corresponding calculation with the roles of x and y interchanged, and y to be the main variable otherwise.

The GCDHEU Implementation

The implementation of the heuristic GCD algorithm in Maple has been improved several times since it was detailed by Char et al. in [3]. There, Char et al. evaluate $y = \alpha$ where α is computed to be the quantity $2\min(||a||_\infty, ||b||_\infty)$. This is followed by removing the integer content from $a(x,\alpha)$ and $b(x,\alpha)$ and computing recursively the GCD of $\mathrm{pp}(a(x,\alpha))$ and $\mathrm{pp}(b(x,\alpha))$. Next the algorithm evaluates at $x = \beta$ where β is chosen to be $2\min(||\mathrm{pp}(a(x,\alpha))||_\infty, ||\mathrm{pp}(b(x,\alpha))||_\infty)$. Since it is likely that $\alpha >> 2||g(x,y)||_\infty$ and $\beta >> 2||pp(g(x,\alpha))||_\infty$, the heuristic should, and from experience does, succeed.

An idea, first suggested by James Davenport, is to first try to show that a and b are relatively prime by using a much smaller evaluation point. The evaluation point must be sufficiently greater than any integer root of $g(x,y)$ to avoid the possibility of returning a proper factor of g.

Another idea is to assume that most likely one of $g, \bar{a}$ or $\bar{b}$ will have height less than $\min(\sqrt{||a||_\infty}, \sqrt{||b||_\infty})$ and to lift the GCD and the cofactors. This means that evaluation points of roughly half the previous size are used at each recursive application of the algorithm.

The version of GCDHEU in Maple V Release 5 applies Davenport's idea once. If this fails, evaluation points based on the minimum of the square root of the heights of a and b are used. Maple attempts to reconstruct both the GCD and the cofactors. If this fails, the implementation tries two slightly larger evaluation points and gives up if both of those attempts are unsuccessful. The implementation also predicts the size of the integers that will be created before evaluation. If they would be larger than 8000 decimal digits, the implementation gives up. Thus the implementation of GCDHEU in Maple should do relatively well on the balanced GCD problems because it will be using a very good estimate for the height of the GCD g, but it should do relatively poorly on the small GCD problems because the estimate for the height of the GCD g will be much too large in this case.

Timing Results

All timings were made on an SGI machine at the CECM at Simon Fraser running Maple V Release 5.0. All times reported are CPU seconds as given by the `time` command in Maple. From the asymptotic results in the previous session, the cost of the two trial divisions in our modified version of Brown's algorithm is expected to dominate the cost of the algorithm for the dense balanced GCD problems. For this reason we have reported this time separately in the column $D(n)$ and have not included in the DIVBRO time. Thus the reader can see how the GCD algorithms compare with polynomial division.

The Balanced GCD Problem

This first set of timings is for dense polynomials. The test polynomials were created as follows.

```
> c := rand(10^n):
> cA := randpoly([x,y],coeffs=c,dense,degree=n):
> cB := randpoly([x,y],coeffs=c,dense,degree=n):
> G := randpoly([x,y],coeffs=c,dense,degree=n):
> A := expand(cA*G):  B := expand(cB*G):
```

The dense option to the `randpoly` command means all terms of total degree $\leq n$ are present. The timing results in Table 3 show that the bottleneck of the new implementation of Brown's algorithm is indeed the two trial divisions.

The second set of timings is for semi-dense polynomials where $\bar{a}, \bar{b}, g$ have $n+3$ terms instead of $O(n^2)$ terms. The test polynomials were created as follows.

```
> c := rand(10^n):
> cA := c()*x^n + c()*y^n + c() +
  randpoly([x,y],coeffs=c,terms=n,degree=n-1):
```

n	$D(n)$	DIVBRO	GCDHEU	EEZGCD
5	.010	.030	.036	0.119
10	.058	.081	1.324	1.99
15	.167	.194	12.85	89.58
20	1.94	.394	74.88	1016.5
25	5.05	.726		
30	12.81	1.50		
35	40.20	2.21		
40	165.6	3.10		
45	767.5	4.67		

Table 3: Dense Balanced GCD

```
> cB := c()*x^n + c()*y^n + c() +
  randpoly([x,y],coeffs=c,terms=n,degree=n-1):
> G  := c()*x^n + c()*y^n + c() +
  randpoly([x,y],coeffs=c,terms=n,degree=n-1):
> A := expand(cA*G):  B := expand(cB*G):
```

The timings in Table 4 confirm that our implementation of Brown's algorithm and the division algorithm are both $O(n^4)$. Note that the polynomials have to be very sparse before the EEZGCD implementation beats DIVBRO.

n	$D(n)$	DIVBRO	GCDHEU	EEZGCD
10	.008	.060	1.055	.181
20	.034	.259	69.144	1.188
30	.175	.761	704.33	21.517
40	1.16	1.780		
50	0.63	4.095		
60	1.96	7.601		
70	5.81	13.181		
80	11.9	21.730		
90	24.3	33.976		
100	41.9	52.561		

Table 4: Semi-Sparse Balanced GCD

The Small GCD Problem

We consider two cases, firstly dense cofactors. The test polynomials were created as follows.

```
> c := rand(10^(N)):
> cA := randpoly([x,y],coeffs=c,dense,degree=N):
> cB := randpoly([x,y],coeffs=c,dense,degree=N):
> c := rand(10^2):
> G := randpoly([x,y],coeffs=c,dense,degree=2);
> A := expand(cA*G):  B := expand(cB*G):
```

The GCD G has coefficients with two decimal digits thus one prime is sufficient to obtain G but two are used in the new algorithm. The timings shown in Table 5 show how badly GCDHEU performs.

In the second case the cofactors are semi-sparse.

```
> c := rand(10^n):
> cA := c()*x^n + c()*y^n + c() +
  randpoly([x,y],coeffs=c,terms=n,degree=n-1):
> cB := c()*x^n + c()*y^n + c() +
  randpoly([x,y],coeffs=c,terms=n,degree=n-1):
> c := rand(10^2):
> G := randpoly([x,y],coeffs=c,dense,degree=2);
> A := expand(cA*G):  B := expand(cB*G):
```

n	$D(n)$	DIVBRO	GCDHEU	EEZGCD
10	.010	.016	.091	.093
20	.036	.025	2.29	.222
30	.090	.046	18.9	.707
40	.154	.071	88.8	5.34
50	.286	.266	331.8	10.0
60	2.46	.157	925.4	21.6
70	1.68	.218		40.1
80	3.35	.294		214
90	20.5	.399		238
100	35.7	.527		557

Table 5: Dense Small GCD

n	$D(n)$	DIVBRO	GCDHEU	EEZGCD
10	.002	.013	.068	.068
20	.006	.016	1.88	.075
30	.012	.024	13.5	.155
40	.017	.033	77.1	.387
50	.026	.045	236.4	.747
80	.046	.078		3.36
120	.104	.147		5.50
160	.172	.254		29.7
200	.925	.481		60.9

Table 6: Semi-sparse Small GCD

This is the best case for the new code in comparison with GCDHEU where the evaluation points are too large. The timings in Table 6 show again how badly GCDHEU performs but also, even though this is the best case for the EEZGCD algorithm, DIVBRO is still much better.

Computing Contents

Some GCD algorithms begin by computing and dividing out the contents in the main variable. This means that two or more recursive GCD computations in one fewer variables over $\mathbb{Z}$ are made. For example, the Maple implementation of the EEZGCD algorithm does this. For some problems the cost of these content calculations is significant. In particular if $g = 1$ but the contents are not 1, these content calculations are wasted. Brown's dense modular GCD algorithm computes the contents one prime at a time at the same time as the primitive part of the GCD is being computed. The timings in Table 7 show that this content computation done by the EEZGCD algorithm can be very significant. More importantly, it means that the complexity of the EEZGCD algorithm is not always good even when $g = 1$. The GCDHEU timings are good because of the small evaluation points being used. But even so, the integers get large enough to cause the GCDHEU to fail on the higher degree problems.

```
> m := iquo(n,3);  c := rand(10^m);  G := 1;
> A := randpoly(x,coeffs=c,dense,degree=m)*
       randpoly(y,coeffs=c,dense,degree=m)*
       (c()*x^m + c()*y^m + c() +
  randpoly([x,y],coeffs=c,terms=m,degree=m-1));
> B := randpoly(x,coeffs=c,dense,degree=m)*
       randpoly(y,coeffs=c,dense,degree=m)*
       (c()*x^M + c()*y^M + c() +
  randpoly([x,y],coeffs=c,terms=M,degree=M-1));
```

n	DIVBRO	GCDHEU	EEZGCD
10	.006	.003	0.024
20	.005	.012	0.052
30	.021	.039	0.157
40	.021	.086	0.268
50	.030	.164	1.130
60	.050	.170	5.396
70	.067	.546	1.093
80	.099	$*$ .437	5.516
90	.122	$*$ 1.074	20.04
100	.144	$*$ .797	71.53

Table 7: Content GCD (* indicates failure)

6. GCDs IN $\mathbb{Q}(\alpha)[X, Y]$

In this section we investigate the computation of GCDs over a number field $\mathbb{Q}(\alpha)$ using our improved version of Brown's algorithm. We again restrict our attention to bivariate polynomials. Let $m(\alpha) \in \mathbb{Z}[\alpha]$ be the minimal polynomial for the number field $\mathbb{Q}(\alpha)$ and let $k = \deg_\alpha m$. The computation of univariate GCDs over number fields using the modular method has been investigated by Langemyr and McCallum [8] and Encarnacion [5]. Encarnacion uses the modular mapping to map elements in $\mathbb{Z}[\alpha]/m(\alpha)$ to $\mathbb{Z}_p[\alpha]/m(\alpha)$ for primes $p = p_0, p_1, \ldots$, computes the modular GCDs over these finite rings and employs the Chinese remainder algorithm and rational reconstruction to recover the rational coefficients of the monic GCD in $\mathbb{Q}[\alpha][x]$. If multiplication and division in the finite ring $\mathbb{Z}[\alpha]/m(\alpha)$ cost $O(k^2)$, then the cost of one univariate GCD of degree n over this ring is $O(n^2k^2)$. The same basic approach can be extended to bivariate polynomials. For the dense balanced GCD problem, the cost of one modular GCD will be $O(n^3k^2)$.

In many practical problems involving number fields, the number field will be of low degree often involving one or two square-roots or cube-roots. In such cases it is not difficult to find primes such that $m(\alpha) \equiv \Pi_{j=1}^{k}(\alpha - \beta_j) \bmod p$, that is, $m(\alpha)$ splits into linear factors. For example, consider $\mathbb{Q}(\sqrt{2}, \sqrt{3})$. Here $m(\alpha) = \alpha^4 - 10\alpha^2 + 1$ and one prime in four will result in $m(\alpha)$ splitting into linear factors. One can then replace the GCD computation over $\mathbb{Z}_p[\alpha]/m(\alpha)$ which costs $O(k^2n^3)$ with k GCD computations over $\mathbb{Z}_p$ which cost $O(kn^3)$ in total. Of course this is only possible when k is small. The gain, in our example, would, theoretically, be a factor of 4, but one is also replacing low degree polynomial computations in $\mathbb{Z}[\alpha]/m(\alpha)$ with computations in $\mathbb{Z}_p$. This is similar to replacing small multi-precision integer operations which involve storage management with machine arithmetic. The following experiment shows that in addition to an expected factor of 16 slowdown in going from $\mathbb{Z}_p$ to $R = \mathbb{Z}_p[\alpha]/(\alpha^4 - 10\alpha^2 + 1)$, a further slowdown of a factor of 22 occurs in Maple. Here $t1$ is the time for a large GCD over $\mathbb{Z}_p$ and $t2$ is the time for a large GCD over R.

```
> p := 46327:
> A := Randpoly(300,x) mod p:
> B := Randpoly(300,x) mod p:
> t1 := time( Gcd(A,B) mod p ):
> rof := RootOf(x^4-10*x^2+1):
> A := Randpoly(300,x,rof) mod p:
> B := Randpoly(300,x,rof) mod p:
```

```
> t2 := time( Gcd(A,B) mod p ):
> t2/(16*t1);
                              22.20192308
```

On the other hand, if the number field has relatively high degree, this overhead drops. For a similar example with the degree 16 minimal polynomial $x^{16} - 136x^{14} + 6476x^{12} - 141912x^{10} + 1513334x^{8} - 7453176x^{6} + 13950764x^{4} - 5596840x^{2} + 46225$ we obtained only a factor of 3.3 slowdown more than the expected factor of 256. Therefore, for number fields, an implementation of Brown's algorithm should map $\mathbb{Z}[\alpha]/(m(\alpha))$ into $\mathbb{Z}_p[\alpha]/m(\alpha)$ for high degree and map $\mathbb{Z}_p[\alpha]/m(\alpha)$ into $\mathbb{Z}_p$ for low degree. We have found that the additional cost of searching for primes such that $m(\alpha)$ has k distinct roots in $\mathbb{Z}_p$ to be negligible for $k \leq 4$.

Included in the comparison timings below is the heuristic method of Gonnet et al. [6] which works by choosing an integer s and computing the GCD of $a(\alpha = s)(x, y)$ and $b(\alpha = s)(x, y)$. The computation is performed modulo an integer q (possibly composite) that divides the integer $m(\alpha = s)$, where q is sufficiently large to enable reconstruction of $g(\alpha)$ from a single image. The GCD computation in $\mathbb{Z}_q[x, y]$ in Maple done using sparse Hensel lifting.

The timings in Table 8 are for the balanced GCD problem over $\mathbb{Q}(\sqrt{2}, \sqrt{3})$ where $m(\alpha) = \alpha^4 - 10\alpha + 1$. The input polynomials were created as follows.

```
> rof := RootOf(x^4-10*x^2+1);
> c := rand(10^n):
> cA := c()*x^n + c()*y^n + c() +
  randpoly([x,y,rof],coeffs=c,terms=n,degree=n-1):
> cB := c()*x^n + c()*y^n + c() +
  randpoly([x,y,rof],coeffs=c,terms=n,degree=n-1):
> G := c()*x^n + c()*y^n + c() +
  randpoly([x,y,rof],coeffs=c,terms=n,degree=n-1):
> A := evala(Expand(cA*G)): B := evala(Expand(cB*G)):
```

n	Heuristic	DIVBRO over $\mathbb{Z}_p$	DIVBRO $\mathbb{Z}_p[\alpha]/m(\alpha)$
5	0.32	0.13	0.70
10	1.94	0.34	5.77
15	10.61	0.91	25.88
20	47.00	1.70	68.41
25	105.53	2.94	153.86
30	294.31	4.93	
35	713.49	8.46	
40		14.42	
45		21.96	
50		43.02	

Table 8: Semi-sparse Balanced GCD over $\mathbb{Q}(\alpha)$

The following set of timings in Table 9 are for the same number field but for the balanced GCD problem with a dense GCD and cofactors. We have included the percentage of time spent performing the divisions in parentheses. The input polynomials were created as follows.

```
> rof := RootOf(x^4-10*x^2+1);
> c := rand(10^n):
> cA := randpoly([x,y,rof],coeffs=c,dense,degree=n):
> cB := randpoly([x,y,rof],coeffs=c,dense,degree=n):
> G := randpoly([x,y,rof],coeffs=c,dense,degree=n):
> A := evala(Expand(cA*G)): B := evala(Expand(cB*G)):
```

n	Heuristic	DIVBRO over $\mathbb{Z}_p$	DIVBRO $\mathbb{Z}_p[\alpha]/m(\alpha)$
5	0.84	0.26 (34%)	0.99
10	23.70	1.93 (70%)	10.26
15	279.88	9.00 (84%)	52.80
20	642.68	33.7 (89%)	235.8

Table 9: Dense Balanced GCD over $\mathbb{Q}(\alpha)$

This final set of timings in Table 10 are for the same number field but for the small GCD problem with semi-sparse cofactors. In this example the Heuristic method does poorly because it does not know that the integer coefficients in g are small.

```
> rof := RootOf(x^4-10*x^2+1);
> c := rand(10^(2*n)):
> cA := c()*x^n + c()*y^n + c() +
  randpoly([x,y,rof],coeffs=c,terms=n,degree=n-1):
> cB := c()*x^n + c()*y^n + c() +
  randpoly([x,y,rof],coeffs=c,terms=n,degree=n-1):
> c := rand(100):
> G := randpoly([x,y,rof],coeffs=c,dense,degree=2):
> A := eval(Expand(cA*G)):  B := eval(Expand(cB*G)):
```

n	Heuristic	DIVBRO over $\mathbb{Z}_p$	DIVBRO $\mathbb{Z}_p[\alpha]/m(\alpha)$
10	0.38	0.12	0.34
20	1.7	0.18	0.94
30	6.2	0.30	1.50
40	18.1	0.70	2.80
50	17.1	0.65	4.26
60	48.2	0.87	6.27
70	65.1	1.00	6.83
80	68.7	1.36	8.75
90	176.2	2.06	11.46
100	–	2.15	15.26

Table 10: Semi-Sparse Small GCD over $\mathbb{Q}(\alpha)$

The timing results show that when using Brown's algorithm, it is worthwhile to map a GCD computation over a small number field into $\mathbb{Z}_p$ instead of $\mathbb{Z}_p[\alpha]/m(\alpha)$.

7. CONCLUSION

We have designed and implemented a variation of Brown's modular GCD algorithm which improves significantly the asymptotic complexity of the small GCD case which occurs frequently in practice. We have also shown that for bivariate polynomials, this algorithm is considerably better asymptotically than the GCDHEU algorithm and the EEZGCD algorithm for dense and semi-sparse polynomials, having the same complexity as polynomial division in each case. Our implementation timings confirm this asymptotic improvement. Despite the simplicity of the GCDHEU algorithm, which maps a GCD computation in $\mathbb{Z}[x,y]$ to a single integer GCD computation, the improved version of Brown's algorithm is faster in practice in almost all cases.

Since in practice GCD problems in a few variables (one, two, or three) are not very sparse, we think that this is the best algorithm to be using in a general purpose CAS.

Our implementation of the improved Brown's algorithm over $\mathbb{Q}(\alpha)$ using the approach taken in the literature is inefficient when the degree of the number field is small. In this case one can obtain an efficient algorithm if one maps into $\mathbb{Z}_p$ by choosing primes for which $m(\alpha)$ splits. This yields an algorithm which is much faster than the Heuristic algorithm of Gonnet et al. in practice on dense and semi-sparse bivariate polynomials.

Note, we did not include our implementation of Zippel's sparse modular GCD algorithm [12] in this comparison as it is not possible at this point to implement it efficiently in Maple (Maple lacks efficient linear algebra facilities for $\mathbb{Z}_p$) to make a fair empirical comparison.

Acknowledgement

We gratefully acknowledge the contribution of George Labahn. The DIVBRO implementation for $\mathbb{Z}[x,y]$ in this paper is replacing an older implementation of Brown's algorithm in Maple by Dr. Labahn which we have studied. That implementation does not minimize the number of primes and evaluation points needed.

8. REFERENCES

[1] L. Bernardin, On Bivariate Hensel Lifting and its Parallelization, *Proceedings of ISSAC '98*, ACM Press (1998), pp. 96–100.

[2] W. S. Brown, On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors, *J. ACM* **18** (1971), pp. 476–504.

[3] B. W. Char, K. O. Geddes, and G. H. Gonnet, GCDHEU: Heuristic polynomial GCD algorithm based on integer GCD computation. *J. Symbolic Comp.* **7** (1989), pp. 31–48.

[4] E. S. Cheb-Terrab and A. D. Roche, Abel Equations: Equivalence and New Integrable Cases *to appear in Comp. Phys. Communications*, June/2000.

[5] M. J. Encarnacion, Computing GCDs of Polynomials over Algebraic Number Fields, *J. Symbolic Computation* **20** (1995), pp. 299–313.

[6] K. O. Geddes, G. H. Gonnet, and T. J. Smedley, Heuristic Methods for Operations with Algebraic Numbers. *Proceedings of ISSAC '88*, Springer-Verlag LNCS **358** (1988), pp. 475–480.

[7] E. Kaltofen and M. B. Monagan, On the Genericity of the Modular Polynomial GCD Algorithm. *Proceedings of ISSAC '99*, ACM Press (1999), pp. 59–66.

[8] L. Langemyr and S. McCallum, The Computation of Polynomial GCD's over an Algebraic Number Field, *J. Symbolic Computation* **8** (1989), pp. 429–448.

[9] A. Miola and D. D. Y. Yun, Computational Aspects of Hensel-type Univariate Polynomial Greatest Common Divisor Algorithms, *Proceedings of EUROSAM '74* (1974), pp. 46–54.

[10] M. B. Monagan, In-place arithmetic for polynomials over $\mathbf{Z}_n$, *Proceedings of DISCO '92*, Springer-Verlag LNCS **721** (1993), pp. 22–34.

[11] P. S. Wang, The EEZ-GCD Algorithm. *ACM SIGSAM Bulletin* **14** (1980), pp. 50–60.

[12] R. Zippel, Probabilistic Algorithms for Sparse Polynomials, *Proc. EUROSAM '79*, Springer-Verlag LNCS **72** (1979), pp. 216–226.