Extended gcd and Hermite normal form algorithms via lattice basis reduction

George Havas
School of Information Technology
The University of Queensland
Queensland 4072, Australia
URL: http://www.it.uq.edu.au/personal/havas/

Bohdan S. Majewski
Department of Computer Science and Software Engineering
University of Newcastle
Callaghan, NSW 2308, Australia
URL: http://wwwcs.newcastle.edu.au/Staff/bohdan/

Keith R. Matthews
Department of Mathematics
The University of Queensland
Queensland 4072, Australia
URL: http://www.maths.uq.edu.au/∼krm/

Abstract

Extended gcd calculation has a long history and plays an important role in computational number theory and linear algebra. Recent results have shown that finding optimal multipliers in extended gcd calculations is difficult. We present an algorithm which uses lattice basis reduction to produce small integer multipliers $x_1, \ldots, x_m$ for the equation $d = \gcd(d_1, \ldots, d_m) = x_1 d_1 + \cdots + x_m d_m$, where $d_1, \ldots, d_m$ are given integers. The method generalises to produce small unimodular transformation matrices for computing the Hermite normal form of an integer matrix.

1 Introduction

Let $d_1, \ldots, d_m$ be integers and $d = \gcd(d_1, \ldots, d_m)$. Then it is easy to find integer multipliers $x_1, \ldots, x_m$ such that $d = x_1 d_1 + \cdots + x_m d_m$, but not so easy to find multiplier vectors X of small Euclidean length $||X|| = (x_1^2 + \cdots + x_m^2)^{1/2}$ (see [17, 22]). Such multipliers may be found by performing, for example, Euclid's algorithm on d_1, d_2, to get $\gcd(d_1, d_2) = g_2$, then on g_2, d_3 and so on. If the corresponding sequence of integer row operations is performed on the identity matrix I_m, the result will be an $m \times m$ unimodular matrix P such that $PD = [0, \ldots, 0, d]^T$, where $D = [d_1, \ldots, d_m]^T$. Such a P is implicit in a paper of Jacobi ([14, pages 26–28]). For some variations on this theme see Kertzner [15], Ford–Havas [8] (who guarantee $|x_i| \leq \max(d_1, \ldots, d_m)/2$) and Majewski–Havas [18] (the sorting gcd algorithm). Also see Brentjes [4, pages 22-24] for references to older work.

With a little matrix algebra, the equation $PD = [0, \ldots, 0, d]^T$ tells us that rows $p_1, \ldots, p_{m-1}$ of P form a lattice basis for the $(m-1)$-dimensional lattice Λ formed by the vectors $X = (x_1, \ldots, x_m)$ with $x_1, \ldots, x_m \in \mathbf{Z}$, satisfying $d_1x_1 + \cdots + d_mx_m = 0$. In other words, every such X can be expressed as an integer linear combination $X = z_1p_1 + \cdots + z_{m-1}p_{m-1}$. The general multiplier vector is given by

$$p_m + z_1p_1 + \cdots + z_{m-1}p_{m-1}, \quad z_1, \ldots, z_{m-1} \in \mathbf{Z}.$$

Lattice basis reduction can be used to find good multipliers. Such an approach dates back at least to Rosser [21] and Ficken [7], who used it for some small examples. A particularly effective algorithm for lattice basis reduction is due to Lenstra, Lenstra and Lovász [16]. For descriptions of the LLL algorithm, see Section 2 and Grötschel et al [10, pages 139–150], Sims [25, pages 360–382], Cohen [5, pages 83–104] or Pohst–Zassenhaus [20, pages 200–202]. Of importance in the LLL algorithm is a parameter α, which is in the range $(\frac{1}{4}, 1]$. The complexity of the algorithm increases with α, as does the quality guarantee on the basis vectors.

One approach to the extended gcd problem, which is proposed by Babai [10, page 144] and Sims [25, page 381], is to perform the LLL algorithm on $p_1, \ldots, p_{m-1}$ to produce a lattice basis of short vectors. We then size–reduce p_m, by adding suitable multiples of these short vectors to p_m, thereby reducing its entries in practice to small size. We call this Algorithm 1. It has the drawback that an initial unimodular transforming matrix P has to be calculated.

Another approach to the problem is to apply the LLL algorithm to the lattice L spanned by the rows of the matrix $C = [I_m|\gamma D]$, where γ is a positive integer. It is not difficult to show that if $\gamma > y^{\frac{m-2}{2}}||D||$, with $y = 4/(4\alpha - 1)$, $1/4 < \alpha \leq 1$, the reduced basis for C must have $c_{1\,m+1} = 0, \ldots, c_{m-1\,m+1} = 0$ and $c_{m\,m+1} = \pm\gamma d$. Then $c_{m1}, \ldots, c_{mm}$ will in practice be a small multiplier vector of similar size to that produced by Algorithm 1. We call this Algorithm 2.

Algorithm 2 works for the following reasons. L consists of the vectors

$$(X, a) = (x_1, \ldots, x_m, \gamma(d_1x_1 + \cdots + d_mx_m)),$$

where $x_1, \ldots, x_m \in \mathbf{Z}$. Hence $X \in \Lambda \Leftrightarrow (X, 0) \in L$. Also, if $(X, a) \in L$ and X does not belong to Λ, then $a \neq 0$ and

$$||(X, a)||^2 \geq \gamma^2. \tag{1}$$

Further, the lemma of [20, page 200] implies that if $b_1, \ldots, b_{m-1}$ form a reduced basis for L, then

$$||b_j|| \leq y^{\frac{m-1}{2}} \max(||X_1||, \ldots, ||X_{m-1}||), \tag{2}$$

if $X_1, \ldots, X_{m-1}$ are linearly independent vectors in L.

But the $m-1$ vectors $X_1, \ldots, X_{m-1}$

$$(-d_2, d_1, 0, \ldots, 0, 0), (-d_3, 0, d_1, 0, \ldots, 0, 0), \ldots, (-d_m, 0, , 0, \ldots, d_1, 0)$$

are linearly independent vectors in L and we have $||X_i|| \leq ||D||$ and hence

$$\max(||X_1||, \ldots, ||X_{m-1}||) \leq ||D||. \tag{3}$$

Hence if $\gamma > y^{\frac{m-2}{2}}||D||$, it follows from inequalities (1)–(3) that the first $m-1$ rows of a reduced basis for L have the form $(b_{j1}, \ldots, b_{jm}, 0)$.

The last vector of the reduced basis then has the form $(b_{m1}, \ldots, b_{mm}, \gamma g)$ for some g, and the equations

$$PD = \begin{bmatrix} 0 \\ g \end{bmatrix}, \quad D = P^{-1} \begin{bmatrix} 0 \\ g \end{bmatrix},$$

(where P is a unimodular matrix) imply $d|g$ and $g|d$, respectively, and hence $g = \pm d$.

Experimentally one finds that if γ is large, Algorithm 2 seems to settle down to the same sequence of row operations. It is not difficult to identify these operations and perform them instead on the matrix $[I_m|D]$. This is justified in Section 3.

Our limiting algorithm is called Algorithm 3 and is described explicitly in Section 4.

In Section 5, we show that with $3/8 < \alpha \leq 1$ the smallest multiplier for 3 numbers is one of the 9 values $b_3 + \epsilon_1 b_1 + \epsilon_2 b_2, |\epsilon_i| \leq 1$, where multiplier b_3 and lattice basis b_1, b_2 for Λ are produced by Algorithm 3. (Computer evidence suggests that the result is true for $1/4 < \alpha \leq 1$.) We also derive an upper estimate in the general case of m numbers for the length of the multiplier produced by Algorithm 3 with $1/4 < \alpha \leq 1$.

In Section 6, we describe a LLL based Hermite normal form algorithm which we also arrive at by limiting considerations.

The paper finishes with some examples which show how well the algorithms perform in practice.

2 The LLL algorithm

In order to analyse Algorithm 2 as $\gamma \to \infty$, we need to briefly outline the LLL algorithm.

Let C be an $m \times n$ matrix of integers, with linearly independent rows $c_1, \ldots, c_m$. The Gram–Schmidt basis is denoted by $c_1^*, \ldots, c_m^*$, where

$$c_1^* = c_1, \quad c_k^* = c_k - \sum_{j=1}^{k-1} \mu_{kj} c_j^*, \quad \mu_{kj} = \frac{c_k \cdot c_j^*}{c_j^* \cdot c_j^*}.$$

We say $c_1, \ldots, c_m$ is *reduced* if $|\mu_{kj}| \leq 1/2$ for $1 \leq j < k \leq m$ and

$$c_k^* \cdot c_k^* \geq (\alpha - \mu_{k\,k-1}^2) c_{k-1}^* \cdot c_{k-1}^* \qquad \text{(C2)}$$

for $1 < k \leq m$. (Here $1/4 < \alpha \leq 1$.) We say c_k is *size–reduced* if $|\mu_{kj}| \leq 1/2$ for $1 \leq j < k$.

The inductive step is as follows:

Do a partial size–reduction by $c_k \leftarrow c_k - \lceil \mu_{k\,k-1} \rfloor c_{k-1}$, where $\lceil \theta \rfloor$ is the nearest integer symbol, with $\lceil \theta \rfloor = \theta - \frac{1}{2}$, if θ is a half–integer. If inequality (C2) holds, size–reduce c_k completely by performing $c_k \leftarrow c_k - \lceil \mu_{kj} \rfloor c_j$ for $j = k-2, \ldots, 1$ and increment k. Otherwise swap c_k and c_{k-1} and decrement k.

3 Analysis of Algorithm 2

Here we justify our earlier assertion that if γ is sufficiently large and the LLL algorithm is performed on $[I_m|\gamma D]$, then the sequence of operations is independent of γ.

Let $c_{i\,m+1} = \gamma a_i$ and let $C = [B|\gamma A]$, where initially $B = I_m, A = D$.

Let us assume that $a_1 = 0, \ldots, a_{k-2} = 0$ and examine the inductive step of LLL.

First, from the equation

$$c_r^* = c_r - \sum_{j=1}^{r-1} \mu_{rj} c_j^*, \tag{4}$$

we have $c_{1\,m+1}^* = 0, \ldots, c_{k-2\,m+1}^* = 0$. Also from equation (4), with $r = k-1$, we have $c_{k-1\,m+1}^* = \gamma a_{k-1}$.

Further

$$\mu_{kj} = \frac{c_k \cdot c_j^*}{c_j^* \cdot c_j^*} = \frac{\sum_{q=1}^m c_{kq} c_{jq}^* + \gamma a_k c_{j\,m+1}^*}{\sum_{q=1}^m (c_{jq}^*)^2 + (c_{j\,m+1}^*)^2}. \tag{5}$$

So, $c_{1\,m+1}^* = 0, \ldots, c_{k-2\,m+1}^* = 0$ and equation (5) give

$$\mu_{kj} = \frac{\sum_{q=1}^m c_{kq} c_{jq}^*}{\sum_{q=1}^m (c_{jq}^*)^2} \quad \text{for } j = 1, \ldots, k-2,$$

the Gram–Schmidt coefficient for C with the last column ignored.

Next

$$\mu_{k\,k-1} = \frac{\sum_{q=1}^m c_{kq} c_{k-1\,q}^* + \gamma^2 a_k a_{k-1}}{\sum_{q=1}^m (c_{k-1\,q}^*)^2 + \gamma^2 a_{k-1}^2} \approx \frac{a_k}{a_{k-1}} \text{ as } \gamma \to \infty.$$

Then if $t = \lceil a_k/a_{k-1} \rfloor$, $\lceil \mu_{k\,k-1} \rfloor = t$ if a_k/a_{k-1} is not an odd multiple of $1/2$, or t or $t+1$ otherwise, as $\gamma \to \infty$. Then t or $t+1$ times row $k-1$ is subtracted from row k.

We now discuss the possible interchange of rows $k-1$ and k. This takes place if the inequality (C2) fails to hold. (We note that $\alpha - \mu_{k\,k-1}^2 > 0$ if $\alpha > \frac{1}{4}$.) If $a_{k-1} = 0 = a_k$, then condition (C2) becomes the standard LLL condition involving $\mu_{k\,k-1}$.

If $a_{k-1} = 0$ but $a_k \neq 0$, then $c_{k\,m+1}^* = \gamma a_k$ and condition (C2) will be satisfied for γ large and no interchange of rows takes place.

If $a_{k-1} \neq 0$, then from

$$c_k^* = c_k - \sum_{j=1}^{k-2} \mu_{kj} c_j^* - \mu_{k\,k-1} c_{k-1}^*,$$

we see that, with $c_{j\,m+1}^* = 0, j = 1, \ldots, k-2$ and with the limiting form of $\mu_{k\,k-1} \approx a_k/a_{k-1}$ above, $c_{k\,m+1}^* \approx 0$. Consequently (C2) will not be satisfied for γ large, if $\alpha > \frac{1}{4}$, and an interchange of rows takes place.

The μ_{kj} will, for large γ, be rational functions of γ and, if not constant, will tend to a limit strictly monotonically, thereby resulting in a limiting sequence of row operations. For large γ the LLL algorithm will perform a version of the least–remainder gcd algorithm (LRA) on $a_1 = d_1, a_2 = d_2$, until it arrives at $a_1 = 0, a_2 = g_2 = \gcd(d_1, d_2)$, with (b_{21}, b_{22}) being the shortest multiplier vector for gcd (d_1, d_2). It then eventually performs a version of the LRA on $a_2 = g_2, a_3 = d_3$, punctuated by updating of the first three rows of B, till it arrives at $a_1 = 0, a_2 = 0, a_3 = g_3 = \gcd(g_2, d_3)$, with (b_{31}, b_{32}, b_{33}) being a short multiplier vector for gcd (d_1, d_2, d_3); and so on.

4 Algorithm 3

We are thus led to the final LLL based extended gcd algorithm given by pseudocode in Figure 1. Our implementation is a modification of de Weger's LLL algorithm [9, pages 329–332], with the added simplification that no initial construction of the Gram–Schmidt basis is necessary, as we start with the identity matrix I_m. De Weger works in terms of integers and writes $|b_i^*|^2 = D_i/D_{i-1}$, $D_0 = 1$ and $\lambda_{ij} = D_j \mu_{ij}$.

5 Multiplier estimates

REMARK. Even when $m = 3$, our LLL based gcd algorithm does not always produce the shortest multiplier: in the example $4, 6, 9$, LLL (for all $\frac{1}{4} < \alpha \leq 1$) produces the multiplier $b_3 = (-2, 0, 1)$, whereas the shortest is $b_3 + b_2 + b_1 = (1, 1, -1)$.

After much numerical experiment we were led to the following result:

THEOREM. If B is a unimodular 3×3 integer matrix such that the first 2 rows b_1, b_2 form a LLL–reduced basis for the lattice Λ with $3/8 < \alpha \leq 1$, while b_3 is size–reduced and is a multiplier vector for d_1, d_2, d_3, then the smallest multiplier is one of nine vectors $b_3 + \epsilon_1 b_1 + \epsilon_2 b_2$, where $\epsilon_i = -1, 0, 1$ for $i = 1, 2$. (Computer evidence strongly suggests that the theorem is true if $1/4 < \alpha$.)

PROOF.

$$b_3 = b_3^* + \mu_{32} b_2^* + \mu_{31} b_1^*, \quad b_2 = b_2^* + \mu_{21} b_1^*, \quad \text{where } |\mu_{ij}| \leq \frac{1}{2}.$$

Then if $x, y \in \mathbf{Z}$, recalling that $b_3 + x b_1 + y b_2$ is the general multiplier, we have the following expression for the square of its length:

$$\begin{aligned} f(x, y) &= ||b_3 + x b_1 + y b_2||^2 = ||b_3^* + (x + \mu_{31} + y\mu_{21}))b_1^* + (y + \mu_{32}) b_2^*||^2 \\ &= ||b_3^*||^2 + (x + \mu_{31} + y\mu_{21})^2 ||b_1^*||^2 + (y + \mu_{32})^2 ||b_2^*||^2. \end{aligned}$$

Using de Weger's notation, working in integers, we write

$$\mu_{ij} = \frac{\lambda_{ij}}{D_j}, \ ||b_1^*||^2 = D_1, \ ||b_2^*||^2 = \frac{D_2}{D_1}, \ ||b_3^*||^2 = \frac{D_3}{D_2} = \frac{1}{D_2},$$

where

$$2|\lambda_{ij}| \leq D_j. \tag{6}$$

($D_3 = 1$ here, as $D_3 = (\det B)^2 = 1$. See [16, equation (1.25)]. Also $D_2 = ||D||^2 / \gcd(d_1, d_2, d_3)^2$, though this is not used.)

Then

$$f(x, y) = \frac{1}{D_2} + \frac{(x D_1 + y\lambda_{21} + \lambda_{31})^2}{D_1} + \frac{(D_2 y + \lambda_{32})^2}{D_1 D_2}. \tag{7}$$

INPUT: Positive integers $d_1, \ldots, d_m$;
$B := I_m$;
$D_i := 1, \quad i = 0, \ldots, m$;
$a_i := d_i, \quad i = 1, \ldots, m$;
$m_1 := 3;\ n_1 := 4;\ \ /*\ \alpha = m_1/n_1\ */$
$k := 2$;

```
while k ≤ m {
    Reduce1(k, k − 1);
    if a_{k−1} ≠ 0 or {a_{k−1} = 0 and a_k = 0
        and n_1(D_{k−2} D_k + λ²_{k k−1}) < m_1 D²_{k−1}} {
      Swap(k);
      if k > 2
        k := k − 1;
    }
    else {
      Reduce1(k, i),     i = k − 2, ..., 1;
      k := k + 1;
    }
}
if a_m < 0 {
    a_m := −a_m;
    b_m := −b_m;
}
```

OUTPUT: $a_m = \gcd(d_1, \ldots, d_m)$; small multipliers $b_{m1}, \ldots, b_{m,m}$;

```
Reduce1(k, i)
    if a_i ≠ 0
        q := ⌈a_k / a_i⌋;
    else {
        if 2|λ_{ki}| > D_i
            q := ⌈λ_{ki}/D_i⌋;
        else q := 0;
    }
    if q ≠ 0 {
        a_k := a_k − q a_i;
        b_k := b_k − q b_i;
        λ_{ki} := λ_{ki} − q D_i;
        for j = 1, ..., i − 1
            λ_{kj} := λ_{kj} − q λ_{ij};
    }
```

```
Swap(k)
    a_k ↔ a_{k−1};
    b_k ↔ b_{k−1};
    for j = 1, ..., k − 2
        λ_{kj} ↔ λ_{k−1 j};
    for i = k + 1, ..., m {
        λ_{i k−1} := (λ_{i k−1} λ_{k k−1} + λ_{ik} D_{k−2})/D_{k−1};
        λ_{ik} := (λ_{i k−1} D_k − λ_{ik} λ_{k k−1})/D_{k−1};
        D_{k−1} := (D_{k−2} D_k + λ²_{k k−1})/D_{k−1};
    }
```

Figure 1: Pseudocode for Algorithm 3

The LLL condition (C2) with $k = 2$ and $1/4 < \alpha \leq 1$ gives

$$\begin{aligned}
||b_2^*||^2 &\geq ||b_1||^2(\alpha - \mu_{21}^2) \\
\frac{D_2}{D_1} &\geq D_1(\alpha - \frac{\lambda_{21}^2}{D_1^2}) \\
D_2 &\geq \alpha D_1^2 - \lambda_{21}^2 \geq (\alpha - \frac{1}{4})D_1^2. \qquad (8)
\end{aligned}$$

Assume $f(x, y) \leq f(0, 0)$. Then we prove $|x|, |y| \leq 1$.

From equation (7), we successively deduce

$$\begin{aligned}
\frac{(xD_1 + y\lambda_{21} + \lambda_{31})^2}{D_1} + \frac{(D_2y + \lambda_{32})^2}{D_1D_2} &\leq \frac{\lambda_{31}^2}{D_1} + \frac{\lambda_{32}^2}{D_1D_2} \qquad (9) \\
\frac{(D_2y + \lambda_{32})^2}{D_1D_2} &\leq \frac{\lambda_{31}^2}{D_1} + \frac{\lambda_{32}^2}{D_1D_2} \\
(D_2y + \lambda_{32})^2 &\leq \lambda_{31}^2 D_2 + \lambda_{32}^2 \\
(y + \frac{\lambda_{32}}{D_2})^2 &\leq \frac{\lambda_{31}^2}{D_2} + \frac{\lambda_{32}^2}{D_2^2} \\
&\leq \frac{D_1^2}{4D_2} + \frac{1}{4} < \frac{8D_2}{4D_2} + \frac{1}{4} = \frac{9}{4}, \qquad (10)
\end{aligned}$$

with the last inequality following from inequality (8) with $\alpha > 3/8$.

Hence $|y + \frac{\lambda_{32}}{D_2}| < \frac{3}{2}$ and $|y| < \frac{3}{2} + \frac{|\lambda_{32}|}{D_2} \leq 2$. Hence $|y| \leq 1$.

Expanding (9) gives

$$(xD_1 + y\lambda_{21} + \lambda_{31})^2 + D_2y^2 + 2\lambda_{32}y \leq \lambda_{31}^2.$$

But

$$D_2y^2 + 2\lambda_{32}y = \begin{cases} 0 & \text{if } y = 0, \\ D_2 \pm 2\lambda_{32} \geq 0 & \text{if } y = \pm 1. \end{cases}$$

Hence

$$\begin{aligned}
|xD_1 + y\lambda_{21} + \lambda_{31}| &\leq |\lambda_{31}| \\
|x + y\frac{\lambda_{21}}{D_1} + \frac{\lambda_{31}}{D_1}| &\leq \frac{|\lambda_{31}|}{D_1} \leq \frac{1}{2} \\
|x| &\leq \frac{1}{2} + |y\frac{\lambda_{21}}{D_1} + \frac{\lambda_{31}}{D_1}| \leq \frac{3}{2},
\end{aligned}$$

which implies $|x| \leq 1$.

REMARK. One can be more specific about the optimum multipliers given the signs of $\lambda_{21}, \lambda_{31}, \lambda_{32}$:

λ_{21}	λ_{31}	λ_{32}	Optimum multiplier
$+$	$+$	$+$	$b_3,\ \ b_3 - b_2$
$+$	$-$	$-$	$b_3,\ \ b_3 + b_2$
$-$	$+$	$-$	$b_3,\ \ b_3 + b_2$
$-$	$-$	$+$	$b_3,\ \ b_3 - b_2$
$-$	$-$	$-$	$b_3,\ \ b_3 - b_2,\ b_3 + b_1 + b_2$
$+$	$+$	$-$	$b_3,\ \ b_3 - b_2,\ b_3 - b_1 + b_2$
$-$	$+$	$+$	$b_3,\ \ b_3 + b_2,\ b_3 - b_1 - b_2$
$+$	$-$	$+$	$b_3,\ \ b_3 + b_2,\ b_3 + b_1 - b_2$

COROLLARY. Our LLL extended gcd algorithm is the basis of a practical polynomial–time algorithm for finding an optimal solution to the extended gcd problem for 3 numbers.

PROOF. Apply Algorithm 3 with $\alpha = 1/2$ and then check which of the nine possibilities is optimal.

THEOREM. Let B be a unimodular $m \times m$ integer matrix such that the first $m-1$ rows form a LLL–reduced basis for the lattice Λ with $1/4 < \alpha \leq 1$, while b_m is size–reduced and is a multiplier vector for $d_1, \ldots, d_m$. Then with $y = 4/(4\alpha - 1)$, we have

$$||b_m||^2 \leq 1 + \frac{m-1}{4} \cdot y^{m-2}||D||^2.$$

PROOF.

$$b_m = b_m^* + \sum_{j=1}^{m-1} \mu_{mj} b_j^*.$$

The vector D^T is orthogonal to $b_1, \ldots, b_{m-1}$ and we see from $BD = [0, \ldots, d]^T$ that $b_m^* = \frac{dD^T}{||D||^2}$. Then

$$b_m = \frac{dD^T}{||D||^2} + \sum_{j=1}^{m-1} \mu_{mj} b_j^*, \quad |\mu_{mj}| \leq \frac{1}{2},\ (j = 1, \ldots, m-1).$$

But $||b_i^*|| \leq ||b_i||$. Hence

$$||b_m||^2 = \frac{d^2}{||D||^2} + \frac{1}{4} \sum_{j=1}^{m-1} ||b_j^*||^2 \leq 1 + \frac{1}{4} \sum_{j=1}^{m-1} ||b_j||^2. \tag{11}$$

Now the lemma of [20, page 200] implies that as $b_1, \ldots, b_{m-1}$ form a reduced basis for Λ, then for $1 \leq j \leq m-1$,

$$||b_j|| \leq y^{\frac{m-2}{2}} \max(||X_1||, \ldots, ||X_{m-1}||),$$

if $X_1, \ldots, X_{m-1}$ are linearly independent vectors in Λ.

But the $m-1$ vectors $X_1, \ldots, X_{m-1}$

$$(-d_2, d_1, 0, \ldots, 0), (-d_3, 0, d_1, 0, \ldots, 0), \ldots, (-d_m, 0, , 0, \ldots, d_1)$$

are linearly independent vectors in Λ and we have $||X_i|| \leq ||D||$ and hence $\max(||X_1||, \ldots, ||X_{m-1}||) \leq ||D||$.

Hence $||b_j|| \leq y^{\frac{m-2}{2}}||D||$ for $j = 1, \ldots, m-1$ and inequality (11) gives

$$||b_m||^2 \leq 1 + \frac{1}{4}(m-1)y^{m-2}||D||^2,$$

as required.

6 A LLL based Hermite normal form algorithm

An $m \times n$ integer matrix B is said to be in Hermite normal form if

(i) the first r rows of B are nonzero;

(ii) for $1 \leq i \leq r$, if b_{ij_i} is the first nonzero entry in row i of B, then $j_1 < j_2 < \cdots < j_r$;

(iii) $b_{ij_i} > 0$ for $1 \leq i \leq r$;

(iv) if $1 \leq k < i \leq r$, then $0 \leq b_{kj_i} < b_{ij_i}$.

Let G be an $m \times n$ integer matrix. Then there are various algorithms for finding a unimodular matrix P such that $PG = B$ is in row Hermite normal form. These include those of Kannan–Bachem [25, pages 349–357] and Havas–Majewski [11], which attempt to reduce coefficient explosion during their execution.

By considering the limiting behaviour of the LLL algorithm on the matrix

$$G(\gamma) = [I_m|\gamma^n G_1|\gamma^{n-1}|G_2|\cdots|\gamma G_n]$$

(where G_i is the ith column of G) as $\gamma \to \infty$, we are led to the following LLL based Hermite normal form algorithm in Figure 2, generalizing the earlier gcd case where $n = 1$. (We have omitted `swap(k)` as it is unchanged, but with a new interpretation of a_i.) It is an easy generalization of the argument in Section 1 to show that for large γ, on LLL reducing $G(\gamma)$, the last n columns form a matrix whose rows, starting from the bottom, are in row echelon form, corresponding to the indices $j_1, \ldots, j_r$.

We remark that if a row of G has to be multiplied by -1, there is a necessary adjustment for the λ_{ij}. Hence the function `Minus`(i).

Let C denote the submatrix of B formed by the r nonzero rows and write $P = \begin{bmatrix} Q \\ R \end{bmatrix}$, where Q and R have r and $m - r$ rows, respectively. Then $QB = C$ and $RB = 0$ and the rows of R will form a $\mathbf{Z}$ basis of short vectors for the sublattice $N(G)$ of $\mathbf{Z}^m$ formed by the vectors X satisfying $XG = 0$. The rows of Q are size–reduced with respect to the short lattice basis vectors for $N(G)$.

We give examples in the next section.

7 Examples

We have applied the methods described here to numerous examples, all with excellent performance. Note that there are many papers which study explicit input sets for the extended gcd problem and a number of these are listed in the references of [4] and [17]. We illustrate algorithm performance with a small selection of interesting examples and make some performance comparisons.

Note also that there are many parameters which can affect the performance of LLL lattice basis reduction algorithms (also observed by many others, including [23]). Foremost is the value of α. Smaller values of α tend to give faster execution times but worse multipliers, however this is by no means uniform. Also, the order of input may have an effect.

```
INPUT: An m × n integer matrix G;
B := I_m;
A := G;
D_i := 1,      i = 0, ..., m;
m_1 := 3;  n_1 := 4;   /* α = m_1/n_1 */
k := 2;
while k ≤ m {
    Reduce2(k, k − 1);
    if {col1 ≤ col2 and col1 ≤ n} or {col1 = col2 = n + 1 and n_1(D_{k−2}D_k + λ²_{k k−1}) < m_1 D²_{k−1}} {
        Swap(k);
        if k > 2
            k := k − 1;
    }
    else {
        Reduce2(k, i),      i = k − 2, ..., 1;
        k := k + 1;
    }
}
OUTPUT: A, the Hermite normal form of G; B the corresponding transformation matrix;
```

```
Reduce2(k, i)
    col1 := least j such that a_{i,j} ≠ 0;
    if a_{i,col1} < 0 {
        Minus(i);
        b_i := −b_i;
    }
    else
        col1 := n + 1;
    col2 := least j such that a_{k,j} ≠ 0;
    if a_{k,col2} < 0 {
        Minus(k);
        b_k = −b_k;
    }
    else
        col2 := n + 1;
    if col1 ≤ n
        q := ⌊a_{k,col1} / a_{i,col1}⌋;
    else {
        if 2|λ_{ki}| > D_i
            q := ⌈λ_{ki}/D_i⌋;
        else q := 0;
    }
    if q ≠ 0 {
        a_k := a_k − q a_i;
        b_k := b_k − q b_i;
        λ_{ki} := λ_{ki} − q D_i;
        for j = 1, ..., i − 1
            λ_{kj} := λ_{kj} − q λ_{ij};
    }
```

```
Minus(j)
    for r = 1, ..., m
        for s = 1, ..., r − 1
            if r = j or s = j
                λ_{rs} := −λ_{rs};
```

Figure 2: A LLL based Hermite normal form algorithm

(a) As input to an extended gcd algorithm, take d_1, d_2, d_3, d_4 to be 116085838, 181081878, 314252913, 10346840.

Algorithm 3 produces a final matrix

$$B = \begin{bmatrix} -103 & 146 & -58 & 362 \\ -603 & 13 & 220 & -144 \\ 15 & -1208 & 678 & 381 \\ -88 & 352 & -167 & -101 \end{bmatrix}.$$

The multiplier vector $(-88, 352, -167, -101)$ is the unique multiplier vector of least length. In fact, LLL-based methods give this optimal multiplier vector for all $\alpha \in (1/4, 1]$.

Earlier algorithms which aim to improve on the multipliers do not fare particularly well. Blankinship's algorithm ([1]) gives the multiplier vector (0, 355043097104056, 1, −6213672077130712). The algorithm due to Bradley ([3]) gives (27237259, −17460943, 1, 0). (This shows that Bradley's definition of minimal is not useful.)

(b) Take $d_1, \ldots, d_{10}$ to be 763836, 1066557, 113192, 1785102, 1470060, 3077752, 114793, 3126753, 1997137, 2603018.

Algorithm 3 gives the following multiplier vectors for various values of α. We also give the length–squared for each vector.

α	multiplier vector x										$\|\|x\|\|^2$
1/4	7	−1	−5	−1	−1	0	−4	0	0	0	93
1/3	−1	0	6	−1	−1	1	0	2	−3	0	53
1/2	−3	0	3	0	−1	1	0	1	−4	2	41
2/3	1	−3	2	−1	5	0	1	1	−2	−1	47
3/4	1	−3	2	−1	5	0	1	1	−2	−1	47
1	−1	0	1	−3	1	3	3	−2	−2	2	42

The unique shortest multiplier vector is $(3, -1, 1, 2, -1, -2, -2, -2, 2, 2)$ with length–squared 36.
Other methods give the following results —
Jacobi: $(-14, 5, -2, 3, -1, 2, -4, 0, -2, 0)$, length–squared 259;
recursive gcd: $(1936732230, -1387029291, -1, 0, 0, 0, 0, 0, 0, 0)$;
Kannan-Bachem: $(44537655090, -31896527153, 0, 0, 0, 0, 0, 0, 0, -1)$;
Blankinship: $(3485238369, 1, -23518892995, 0, 0, 0, 0, 0, 0, 0)$;
Bradley: $(-135282, 96885, -1, 0, 0, 0, 0, 0, 0, 0)$.

(c) The following example involving Fibonacci and Lucas numbers (see [13]) has theoretical significance. Take $d_1, \ldots, d_m$ to be the Fibonacci numbers

(i) $F_n, F_{n+1}, \ldots, F_{2n}$, n odd, $n \geq 5$;

(ii) $F_n, F_{n+1}, \ldots, F_{2n-1}$, n even, $n \geq 4$.

Using the identity $F_m L_n = F_{m+n} + (-1)^n F_{m-n}$, it can be shown that the following are multipliers:

(i) $-L_{n-3}, L_{n-4}, \ldots, -L_2, L_1, -1, 1, 0, 0, \quad n$ odd;

(ii) $L_{n-3}, -L_{n-4}, \ldots, -L_2, (L_1+1), -1, 0, 0, \quad n$ even,

where $L_1, L_2, \ldots$ denote the Lucas numbers $1, 3, 4, 7, \ldots$

These multipliers are the unique vectors of least length. (This is a special case of a more general result of the third author [19], where $F_n, \ldots, F_{n+m}$ is treated.) The length–squared of the multipliers is $L_{2n-5}+1$ in both cases. (In practice, the LLL-based algorithms compute these minimal multipliers.)

These results give bounds for extended gcd multipliers in terms of Euclidean norms. Since, with $\phi = \frac{1+\sqrt{5}}{2}$, $L_{2n-5}+1 \sim \phi^{2n-5} \sim \phi^{-5}\sqrt{5}F_{2n}$ it follows that a general upper bound for the Euclidean norm of the multiplier vector in terms of the initial numbers d_i must be at least $O(\sqrt{\max\{d_i\}})$. Also, the length of the vector $(F_n, F_{n+1}, \ldots, F_{2n})$ is of the same order of magnitude as F_{2n}, so a general upper bound for the length of the multipliers in terms of the Euclidean length of the input, l say, is at least $O(\sqrt{l})$.

A range of random type extended gcd examples is presented in [12].

For a Hermite normal form example, take $G = [g_{ij}]$ to be the 10×10 matrix defined by $g_{ij} = i^3 * j^2 + i + j$:

$$
G = \begin{bmatrix}
3 & 7 & 13 & 21 & 31 & 43 & 57 & 73 & 91 & 111 \\
11 & 36 & 77 & 134 & 207 & 296 & 401 & 522 & 659 & 812 \\
31 & 113 & 249 & 439 & 683 & 981 & 1333 & 1739 & 2199 & 2713 \\
69 & 262 & 583 & 1032 & 1609 & 2314 & 3147 & 4108 & 5197 & 6414 \\
131 & 507 & 1133 & 2009 & 3135 & 4511 & 6137 & 8013 & 10139 & 12515 \\
223 & 872 & 1953 & 3466 & 5411 & 7788 & 10597 & 13838 & 17511 & 21616 \\
351 & 1381 & 3097 & 5499 & 8587 & 12361 & 16821 & 21967 & 27799 & 34317 \\
521 & 2058 & 4619 & 8204 & 12813 & 18446 & 25103 & 32784 & 41489 & 51218 \\
739 & 2927 & 6573 & 11677 & 18239 & 26259 & 35737 & 46673 & 59067 & 72919 \\
1011 & 4012 & 9013 & 16014 & 25015 & 36016 & 49017 & 64018 & 81019 & 100020
\end{bmatrix}.
$$

Then the Hermite normal form B of G has three nonzero rows given by

$$
\begin{bmatrix}
1 & 0 & 7 & 22 & 45 & 76 & 115 & 162 & 217 & 280 \\
0 & 1 & 4 & 9 & 16 & 25 & 36 & 49 & 64 & 81 \\
0 & 0 & 12 & 36 & 72 & 120 & 180 & 252 & 336 & 432
\end{bmatrix}.
$$

The unimodular matrix provided by the Kannan–Bachem algorithm is

$$
\begin{bmatrix}
-48 & 47 & -12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-8 & 10 & -5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-62 & 57 & -12 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
982 & -2620 & 2295 & -658 & 1 & 0 & 0 & 0 & 0 & 0 \\
1684 & -4495 & 3940 & -1130 & 0 & 1 & 0 & 0 & 0 & 0 \\
2662 & -7108 & 6233 & -1788 & 0 & 0 & 1 & 0 & 0 & 0 \\
3962 & -10582 & 9282 & -2663 & 0 & 0 & 0 & 1 & 0 & 0 \\
5630 & -15040 & 13195 & -3786 & 0 & 0 & 0 & 0 & 1 & 0 \\
7712 & -20605 & 18080 & -5188 & 0 & 0 & 0 & 0 & 0 & 1 \\
-3 & 8 & -7 & 2 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

whereas that supplied by our algorithm is

$$\begin{bmatrix} -10 & -8 & -5 & 1 & 2 & 3 & 5 & 3 & 0 & -4 \\ -2 & -1 & 0 & 1 & -1 & 0 & 1 & 0 & 1 & -1 \\ -15 & -11 & -4 & 0 & 4 & 5 & 4 & 3 & 1 & -5 \\ 1 & -1 & -1 & 0 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 & 1 & -1 & 2 & -1 & 0 & 0 \\ 1 & 0 & -1 & -1 & -1 & 2 & 0 & 1 & -1 & 0 \\ 1 & 0 & -2 & 1 & -1 & 1 & -1 & 1 & 1 & -1 \\ -1 & 0 & 1 & 0 & 1 & 1 & -1 & -2 & 0 & 1 \\ 1 & -1 & 0 & -1 & 1 & 0 & 0 & -1 & 2 & -1 \\ 1 & -2 & 1 & 1 & -2 & 0 & 2 & -1 & 0 & 0 \end{bmatrix}.$$

An interesting family of matrices arises in the work of Daberkow [6]. In some ideal class group work matrices arise with k rows and 10 columns for k ranging from 100 to 150 in steps of 10. We designate the matrix with k rows by M_k. The maximal magnitude entry in M_k is of the order $11^{(k-90)/10}$. Daberkow needs to compute both the Hermite normal form and a transforming matrix. We tabulate the maximal magnitude entry in the tranforming matrix (which includes many entries of this size) for our algorithm using $\alpha = 1$ in comparison with that of Kannan–Bachem.

	Kannan–Bachem	LLL HNF
M_{100}	77710953119323250210825968427763925730604	2
M_{110}	19688024435949960842280085386879376037295267254	2
M_{120}	32807912677637850341882990335	4
M_{130}	258209178730643422634648900270488370181908068255159901037863837761499488802313834227	5
M_{140}	8877061573605684598479855792299	9
M_{150}	143547860664185870781020896285	9

8 Conclusions

We have presented new algorithms for extended gcd calculation which compute good multipliers. We have provided analyses of their performance. We have given examples which show them dramatically outperforming earlier methods. Related algorithms which compute the Hermite normal form of an integer are presented with examples showing excellent performance.

REMARK. Implementations of these algorithms are available in the third author's number theory calculator program CALC at `http://www.maths.uq.edu.au/~krm/`. Variants of these algorithms are available in GAP ([24]) and MAGMA ([2]).

Acknowledgments

The first two authors were supported by the Australian Research Council. We are grateful to Jean-Pierre Seifert for helpful discussions.

References

[1] W.A. Blankinship, *A new version of the Euclidean algorithm*, Amer. Math. Mon. 70 (1963) 742–745.

[2] W. Bosma and J. Cannon, *Handbook of* MAGMA *functions*, Department of Pure Mathematics, Sydney University, 1996.

[3] G.H. Bradley, *Algorithm and bound for the greatest common divisor of n integers*, Communications of the ACM 13 (1970) 433–436.

[4] A.J. Brentjes, *Multi–dimensional continued fraction algorithms*, Mathematisch Centrum, Amsterdam 1981.

[5] H. Cohen, *A Course in Computational Number Theory*, Graduate Text 138, Springer 1993.

[6] M. Daberkow, *Ueber die Bestimmung der ganzen Elemente in Radikalerweiterungen algebraischer Zahlkoerper*, Dissertation, Tech. Univ. Berlin, Berlin 1995.

[7] F.A. Ficken, *Rosser's generalization of the Euclid algorithm*, Duke Math. J. 10 (1943) 355–379.

[8] D. Ford and G. Havas, *A new algorithm and refined bounds for extended gcd computation*, Algorithmic Number Theory, Lecture Notes in Computer Science 1122, 145–150, Springer 1996

[9] B.M.M de Weger, *Solving exponential Diophantine equations using lattice basis reduction*, J. Number Theory 26 (1987) 325–367.

[10] M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer–Verlag, Berlin 1988.

[11] G. Havas and B.S. Majewski, *Hermite normal form computation for integer matrices*, Congressus Numerantium 105 (1994) 87–96.

[12] G. Havas and B.S. Majewski, *Extended gcd calculation*, Congressus Numerantium 111 (1995) 104–114.

[13] V.E. Hoggatt Jr., *Fibonacci and Lucas Numbers*, Houghton Mifflin Company, Boston 1969.

[14] C.G.J. Jacobi, *Über die Auflösung der Gleichung $\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n = f \cdot u$*, J. Reine Angew. Math. 69 (1868) 1–28.

[15] S. Kertzner, *The linear diophantine equation*, Amer. Math. Monthly 88 (1981) 200–203.

[16] A.K. Lenstra, H.W. Lenstra Jr., and L. Lovász. *Factoring polynomials with rational coefficients*, Math. Ann. 261 (1982) 515–534.

[17] B.S. Majewski and G. Havas, *The complexity of greatest common divisor computations*, Algorithmic Number Theory, Lecture Notes in Computer Science 877, 184–193, Springer 1994.

[18] B.S. Majewski and George Havas, *A solution to the extended gcd problem*, ISSAC'95 (Proc. 1995 International Symposium on Symbolic and Algebraic Computation), ACM Press (1995) 248–253.

[19] K.R. Matthews, *Minimal multipliers for consecutive Fibonacci numbers*, Acta Arith. (1996) 75, 205–218.

[20] M. Pohst and H. Zassenhaus, *Algorithmic Algebraic Number Theory*, Cambridge University Press, 1989.

[21] Barkley Rosser, *A Note on the Linear Diophantine Equation*, Amer. Math. Monthly 48 (1941) 662–666.

[22] C. Rössner and J.–P. Seifert, *The Complexity of Approximate Optima for Greatest Common Divisor Computations*, Algorithmic Number Theory, Lecture Notes in Computer Science 1122, 307–322, Springer 1996

[23] C.P. Schnorr and M. Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*, Lecture Notes in Computer Science 529, 68–85, 1991.

[24] M. Schönert *et al.*, GAP – *Groups, Algorithms and Programming*, Lehrstuhl D für Mathematik, RWTH, Aachen, 1996.

[25] C.C. Sims, *Computing with finitely presented groups*, Cambridge University Press, 1994.