A Solution to the Extended GCD Problem with Applications*

Arne Storjohann

Eidgenössische Technische Hochschule
CH-8092 Zürich, Switzerland
storjoha@inf.ethz.ch
http://www.inf.ethz.ch/personal/storjoha

Abstract

This paper considers a variation of the extended gcd problem: the "modulo N extended gcd problem". Given an integer row vector $[a_i]_{i=1}^n$, the modulo N extended gcd problem asks for an integer vector $[c_i]_{i=1}^n$ such that

$$\gcd(\sum_{i=1}^{n} c_i a_i, N) = \gcd(a_1, a_2, \ldots, a_n, N).$$

A deterministic algorithm is presented which returns an exceptionally small solution for a given instance of the problem: both $\max_{i=1}^n |c_i|$ and the number of nonzero c_i's will be bounded by $O(\log N)$. The gcd algorithm presented here has numerous applications and has already led to faster algorithms for computing row reduced echelon forms of integer matrices and solving systems of linear Diophantine equations. In this paper we show how to apply our gcd algorithm to the problem of computing small pre- and post-multipliers for the Smith normal of an integer matrix.

1 Introduction

This paper considers the following problem: Given a nonnegative integer N together with an integer row vector $a = [a_1, a_2, \ldots, a_n]$, find an integer vector $c = [c_1, c_2, \ldots, c_n]$ such that

$$\gcd(\sum_{i=1}^{n} c_i a_i, N) = \gcd(a_1, a_2, \ldots, a_n, N). \qquad (1)$$

For the special case when N is zero, this is known as the "extended gcd problem". In the case where N is positive, we call this the "modulo N extended gcd problem". In ring theoretic terms, $\sum_{i=1}^n c_i a_i$ should be a generator of the ideal $\langle a_1, a_2, \ldots, a_n \rangle$ in the ring of integers modulo N.

Solutions to the extended gcd and modulo N extended gcd problem are required in such problems as computing canonical triangular and diagonal forms of matrices over the domains the integers [6, 7, 11, 19, 20] and the integers modulo N [10, 21]. In such applications it is often beneficial to get a "good" solution vector c, that is, one which is small with respect to the L_0 metric (which counts the number of nonzero c_i's) or the L_∞ norm (which bounds the magnitude of the largest c_i). The problem of computing good solutions for the extended gcd problem has recently enjoyed considerable attention [2, 8, 9, 15, 16]. In this paper we consider the problem of computing a good solution for the modulo N extended gcd algorithm

The main result of this paper is a deterministic algorithm for solving the modulo N extended gcd problem. The algorithm returns a solution for $c = [c_1, c_2, \ldots, c_n]$ that satisfies the following properties:

($e1$) $c_1 = 1$.
($e2$) At most $\lfloor \log_2 N \rfloor$ of the c_i's will be nonzero.
($e3$) $|c_i| \leq \lceil 2(\log_2 N)^{3/2} \rceil$ for $1 \leq i \leq n$.

If we assume that entries in the input vector a are bounded in magnitude by N, then the cost of the modulo N extended gcd algorithm is $O(n \log^2 N + \log^3 N)$ bit operations assuming standard (quadratic) integer arithmetic. Properties ($e1$) and ($e2$) are relatively easily satisfied. The main contribution of our modulo N extended gcd algorithm is that property ($e3$) will also be satisfied. In particular, ($e3$) ensures that entries in the solution vector c will be bounded in length by $O(\log \log N)$ bits; a typical solution vector, on the other hand, will have entries bounded in length by $O(\log N)$ bits.

Consider the modulo N extended gcd problem with with $N = 223092870$ and input vector

$$a = [56039340, 45020850, 114868782, 145800000].$$

A typical solution vector (i.e. one not returned by our algorithm) for this problem is

$$c = [137521213, 189470769, 155848668, 36654910]$$

which has entries with the same bit length as N. Such a solution vector can be found, for example, by setting c to be a vector with entries chosen uniformly and randomly in the range 0 and $N - 1$, then testing equation (1) for correctness and repeating with a new random choice if required (see [3] for the details of such a Las Vegas probabalistic algorithm). On the other hand, the algorithm presented here returns the solution vector

$$c = [1, 3, 6, 10].$$

In what follows we summarize some new results that have been obtained by applying our modulo N extended gcd algorithm to the problem of computing matrix canonical forms. To give complexity results we use the parameters θ and ϵ.

*This work has been partially supported by grants from the Swiss Federal Office for Education and Sciences in conjunction with partial support by ESPRIT LTR Project no. 20244 — ALCOM-IT.

Two $n \times n$ matrices over a ring can be multiplied in $O(n^\theta)$ ring operations and two $[t]$ bit integers can be multiplied in $O(t^{1+\epsilon})$ bit operations. The asymptotically fast (but currently impractical) algorithm of Coppersmith & Winograd [1] and the pseudo-linear algorithm of Schönhage & Strassen [17] allow $\theta = 2.376$ and any fixed (but positive) ϵ respectively. All algorithms in this paper assume the standard, practical algorithms for integer and matrix multiplication which have $\epsilon = 1$ and $\theta = 3$. In what follows we write $||A||$ to denote the largest magnitude entry of an integer matrix A.

In [20] we apply our modulo N extended gcd algorithm to get a very fast, practical and deterministic algorithm for triangularizing integer matrices. The major breakthrough of this new triangularization algorithm is that it allows integer arithmetic to be performed in a residue number system modulo a basis of word-size primes. Crucial to the method used in [20] is a subroutine for computing very small solutions to the modulo N extended gcd problem; the current paper presents such a solution. The new triangularization algorithm of [20] leads directly to efficient solutions for the following problems: (1) computing the Hermite normal form with transforming matrix for an integer input matrix of arbitrary shape and rank profile, and; (2) computing the general solution to a system of linear Diophantine equations.

A second application of our gcd algorithm is to computing transforming matrices for the Smith normal form of an integer matrix. First recall the definition of the Smith normal form. It follows from Smith [18] that corresponding to any $A \in \mathbf{Z}^{n \times m}$ there exist unimodular (square and invertible) matrices U and V over $\mathbf{Z}$ such that

$$UAV = S = \mathrm{diag}(s_1, s_2, \ldots, s_r, 0, \ldots, 0)$$

with each s_i nonzero and with $s_i | s_{i+1}$ for $1 \leq i \leq r-1$. S is called the Smith normal form of A and the unimodular U and V are called transforming matrices. While the Smith normal form is a unique canonical form, the transforming matrices are highly nonunique and most previous algorithms for computing S don't return transforming matrices (cf. [3, 4, 5, 19, 21]). The previously fastest algorithm [11] for producing a U and V has been presented for the case of a square nonsingular input matrix A and requires $O^\sim(n^{5.376} \log^2 ||A||)$ bit operations[1] — assuming asymptotically fast matrix multiplication and pseudo-linear integer arithmetic — to produce a U and V with entries bounded in length by $O^\sim(n^3 \log^2 ||A||)$ bits. The algorithm we present here requires only $O^\sim(n^5 \log^2 ||A|| + n^3 \log^3 ||A||)$ bit operations — assuming standard integer and matrix multiplication — to produce a U and V with entries bounded in length by $O(n \log ||A||)$ bits. Moreover, the total size of V, the sum of the bit lengths of all the entries, will be bounded by $O^\sim(n^2 \log ||A||)$ bits — this is on the same order of space required to write down the input matrix A. For comparison, the previously fastest algorithm produces a V with worst case size bound $O^\sim(n^5 \log^2 ||A||)$ bits — this bound is a factor of about $O(n^3 \log ||A||)$ larger then the space required to write down to input matrix. The Smith normal form algorithm we present here also recovers transforming matrices for the case of a rectangular input matrices of full column rank.

The rest of this paper is organised as follows. In Section 2 we present some number theoretic algorithms which form the basis of our modulo N extended gcd algorithm. In Section 3 we present the gcd algorithm itself and show how it applies to the problem of computing canonical matrix forms. Section 4 presents the new algorithm for computing pre- and post-multipliers for the Smith normal form. We consider this Smith normal form algorithm to be very practical — some explicit examples demonstrating the good performance of the new algorithm are given in Subsection 4.2. We conclude in Section 5 by indicating some relationships of our Smith normal form algorithm with previous work and also mention some ideas for future research.

[1] To summarize complexity results we use soft-"Oh" notation: for real valued functions f and g, $f = O^\sim(g)$ if and only if $f = O(g \cdot \log^c g)$ for some constant $c > 0$.

2 Some Number Theoretic Algorithm

This section develops an algorithmic solution to the following problem: Given integers a, b, N with N positive and $\gcd(a, b) = 1$, find the smallest nonnegative integer c that satisfies

$$\gcd(a + cb, N) = 1 \tag{2}$$

Our first result establishes a necessary and sufficient condition for c to satisfy (2).

Lemma 1 *Let a,b,N be integers with N positive and $\gcd(a, b) = 1$, and let $\{p_1, p_2, \ldots, p_r\}$ be the set of distinct primes which divide N but not b. An integer c will satisfy $\gcd(a + cb, N) = 1$ if and only if $a + cb \neq 0 \pmod{p_i}$ for $1 \leq i \leq r$.*

Proof. (*Only If:*) Let c satisfy (2). Assume, to arrive at a contradiction, that $a + cb = 0 \pmod{p_i}$ for some i with $1 \leq i \leq r$. Since $p_i | N$ we must have $p_i | \gcd(a + cb, N)$, a contradiction. (*If:*) Let c satisfy $a + cb \neq 0 \pmod{p_i}$ for $1 \leq i \leq r$. Assume, to arrive at a contradiction, that $\gcd(a + cb, N) > 1$ and let p be a prime which divides $\gcd(a + cb, N)$. Then $p | (a + cb)$ and $p | N$. It follows that $a + cb = 0 \pmod{p}$ and by assumption p must be a prime which divides N *and* b. In particular, we must have $b = 0 \pmod{p}$. But then $\gcd(a, b) = 1$ implies p does not divide a which, together with $b = 0 \pmod{p}$, implies $a + cb = a \neq 0 \pmod{p}$, a contradiction since we have already seen $a + cb = 0 \pmod{p}$. ∎

The following fact and subsequent theorem establishes the existence of a small c which satisfies (2).

Fact 1 (Kanold [14]) *Let $1 = a_1 < a_2 < \cdots$ be the sequence of integers relatively prime to a positive integer N and let $g(N) = \max(a_{i+1} - a_i)$. If N has r distinct prime factors, then $g(N) < 2r^{3/2}$.*

Theorem 2 *Let r be a bound on the number of distinct prime divisors of a positive integer N. Given integers a and b with $\gcd(a, b) = 1$, there exists a nonnegative integer c bounded in magnitude by $2r^{3/2}$ that satisfies $\gcd(a+cb, N) = 1$.*

Proof. Let P be the product of all distinct primes dividing N but not b. Let C be the unique integer with $0 \leq C < P$ which satisfies $a - Cb = 0 \pmod{P}$. (Uniqueness follows from the Chinese remainder theorem.) Let U be the integer greater than or equal to C which is relatively prime to N and which minimizes the quantity $U - C$. Then $a + (U - C)b \neq 0 \pmod{p}$ for all primes p dividing N but not b, and by Lemma 1, $\gcd(a + (U - C)b, N) = 1$. Set $c = U - C$. To prove that

c is small enough, let L to be the integer less than or equal to C which is relatively prime to N and which minimizes the quantity $C - L$. The result follows by noting that $c = U - C \leq U - L$, and by Fact 1, $U - L \leq g(N) < 2r^{3/2}$. ■

We now give three deterministic algorithms for computing the quantity c of Theorem 2. Algorithm I, the brute force search approach, admits a worst case complexity of $O(\log^{3.5} N)$ bit operations but has the advantage of simplicity. Algorithm II requires as input a list of distinct prime factors of N and uses a number sieve to compute c in only $O(\log^2 N)$ bit operations. This algorithm is useful for cases where N can be factored easily. Algorithm III combines ideas from Algorithm I and II. In particular, the complexity is better than that of Algorithm I but we don't need as input a list of distinct prime factors of N. Before giving the algorithms, we first recall the complexity model for standard integer arithmetic.

The number of bits in the binary representation of an integer a is given by

$$\lg a = \begin{cases} 1, & \text{if } a = 0; \\ 1 + \lfloor \log_2 |a| \rfloor, & \text{if } |a| > 0. \end{cases}$$

Given integers a, b and M we can: compute the product ab in $O((\lg a)(\lg b))$ bit operations; compute integers q and r such that $a = qb + r$ with $0 \leq |r| < |n|$ in $O((\lg a/b)(\lg b))$ bit operations; compute the gcd g of a and b in in $O((\lg a/g)(\lg b))$ bit operations when $|a| \geq |b|$; compute either direction of the isomorphism implied by the Chinese remainder theorem in $O(\lg^2 M)$ bit operations where M is the product of the modulii.

2.1 Algorithm I: Brute Force

Theorem 3 *Let a, b, N be integers with N positive and $\gcd(a, b) = 1$. There exists a deterministic algorithm that computes the smallest nonnegative integer c that satisfies $\gcd(a + cb, N) = 1$. If $|a|, |b| < N$, then the cost of the algorithm is bounded by $O(\log^{3.5} N)$ bit operations.*

Proof. Compute $\gcd(a + tb, N)$ for $t = 0, 1, 2, \ldots$ and set c to be the minimum t with $\gcd(a + tb, N) = 1$. By Theorem 2, the solution c will be found after at most $\lfloor 2r^{3/2} \rfloor$ steps, where r is the number of distinct prime divisors of N. Each test requires at most $O(\log^2 N)$ bit operations for the gcd computation, and r is bounded by $O((\log N)/(\log \log N))$, leading to a worst case complexity of $O(\log^{3.5} N)$ bit operations. ■

2.2 Algorithm II: Prime Factorization

Theorem 4 *Let a, b, N be integers with N positive and $\gcd(a, b) = 1$. Given the set $\{p_1, p_2, \ldots, p_r\}$ of distinct prime divisors of N, there exists a deterministic algorithm that that computes the smallest nonnegative integer c that satisfies $\gcd(a + cb, N) = 1$. If $|a|, |b| < N$, then the cost of the algorithm is bounded by $O(\log^2 N)$ bit operations.*

Proof. Set $K = \lfloor 2r^{3/2} \rfloor$ and initialize all entries to true in a binary array B indexed from 0 to K. The idea is to sieve out all bad points from the set $\{0, 1, \ldots, K\}$ according the criteria of Lemma 1. For $0 \leq t \leq K$, $B[t]$ = false will indicate that $\gcd(a + tb, N) > 1$. For $i = 1, 2, \ldots, r$, set $B[t]$ equal to false for all integers $0 \leq t \leq K$ that satisfy $a + tb = 0 \pmod{p_i}$. This is accomplished by the following nested loop, where the binary function $\bmod(q, d)$ takes as first argument a rational number, as second argument a positive integer, and returns the unique nonnegative integer congruent to q modulo d and in the range 0 to $p - 1$.

```
for i = 1 to r do
    (a_i, b_i) ← (mod(a, p_i), mod(b, p_i));
    if b_i ≠ 0 then
        s_i ← mod(−a_i/b_i, p_i)
        for j = 0 to ⌊(K − s_i)/p_i⌋
            B[s_i + jp_i] ← false;
```

Now, by Lemma 1, an integer t between 0 and K will satisfy $\gcd(a + tb, N) = 1$ if and only if $B[t]$ is still equal to true, and by Theorem 2, there exists at least one such t with $B[t]$ true. Do a linear search of the array B and set c to be the smallest index t with $B[t]$ true. We now show that the above sieving procedure can be accomplished in $O(\log^2 N)$ bit operations. The computation of all the a_i's, b_i's, s_i's and upper indices $\lfloor (K - s_i)/p_i \rfloor$ of the inner loop is bounded by $O(\log^2 N)$ bit operations since the product of the p_i's is bounded by N. The number of iterations of the inner loop is given by

$$\begin{aligned} \sum_{1 \leq i \leq r,\, b_i \neq 0,\, 0 \leq s_i \leq K} \lfloor (K - s_i)/p_i \rfloor + 1 &\leq r + \sum_{i=1}^{r} K/p_i \\ &\leq r + K \sum_{i=1}^{r} 1/i \\ &\leq r + K(1 + \log r) \\ &< hK \log r \end{aligned}$$

for some absolute constant h. Computing the index of B in the inner loop involves numbers bounded in magnitude by K and so requires $O(\log^2 K)$ bit operations. Since $K = O(r^{3/2})$ where $r = O((\log N)/(\log N \log \log N))$, the total cost of the inner loop is bounded by $O(\log^{3/2} N)$ bit operations, and the entire procedure by $O(\log^2 N)$ bit operations. ■

2.3 Algorithm III: Integer Factor Refinement

The drawback with the algorithm of Theorem 4 is that it requires as input a list of all prime divisors of N. Finding all prime divisors of N is equivalent to computing the complete factorization of N — no efficient algorithm is known for this problem. The algorithm we propose next doesn't require as input the complete factorization of N, but rather takes as input some factorization $[d_1, d_2, \ldots, d_q]$ of N. Here, a list $[d_1, d_2, d_3, \ldots, d_q]$ of integers is a factorization of N of length q if $d_i > 1$ for $1 \leq i \leq q$ and $d_1 d_2 \cdots d_q = N$. Note that the d_i's need not be distinct. Rather, the important point for our purposes is that $d_i > 1$ for $1 \leq i \leq q$. The following algorithm `Conditioner` requires $O(\log^2 N)$ bit operations to produce either: (a) the smallest c satisfying $\gcd(a + cb, N) = 1$, OR (b) a new factorization of N with length greater than q. Since the complete factorization of N can have length at most $\lg N$, this bounds the number of times case (b) can occur.

Algorithm: `Conditioner`
Input: Integers a, b, N with N positive and $\gcd(a, b) = 1$. A factorization $[d_1, d_2, \ldots, d_q]$ of N.
Output: Either the smallest nonnegative integer c satisfying $\gcd(a + cb, N) = 1$ OR a new factorization for N with length greater than q.

(1) [Initialize:]
$K \leftarrow \lfloor 2(\log_2 N)^{3/2} \rfloor$;
for $t = 0$ to K do $B[t] \leftarrow$ true;
(2) [Compute residues:]
for $i = 1$ to q do
$(a_i, b_i) \leftarrow (\mathrm{mod}(a, d_i), \mathrm{mod}(b, d_i))$;
$g_i \leftarrow \gcd(b_i, d_i)$;
if $2 \leq g_i < d_i$ then goto (6);
(3) [Sieve:]
for $i = 1$ to q do
if $b_i \neq 0$ then
$s_i \leftarrow \mathrm{mod}(-a_i/b_i, d_i)$;
for $j = 0$ to $\lfloor (K - s_i)/d_i \rfloor$
$B[s_i + jd_i] \leftarrow$ false
(4) [Find candidate solution and assay correctness:]
$t \leftarrow$ the minimum index with $B[t]$ true;
for $i = 1$ to q do
$g_i \leftarrow \gcd(a_i + tb_i, d_i)$;
if $g_i > 1$ then goto (6);
(5) [Ouput solution:]
return t and quit;
(6) [Ouput nontrivial factor refinement:]
return $[d_1, \ldots, d_{q-1}, g_i, d_i/g_i, d_{q+1}, \ldots, d_q]$ and quit;

Theorem 5 *Algorithm* `Conditioner` *is correct. If* $|a|, |b| < N$ *then the cost of the algorithm is* $O(\log^2 N)$ *bit operations.*

Proof. Step (1) is bounded by $O(\log^{3/2} N)$ bit operations and step (2) by $O(\log^2 N)$ bit operations since $\prod_{1 \leq i \leq q} d_i = N$ and $d_i > 1$ for $1 \leq i \leq q$. In step (2), if some g_i satisfies $2 \leq g_i < d_i$ for $1 \leq i \leq q$, then this provides a nontrivial factorization of d_i in step (6). On the other hand, if step (3) is reached, then $b_i \neq 0 \pmod{d_i}$ implies b_i is relatively prime to d_i; this shows that the computation of $s_i \leftarrow \mathrm{mod}\ (-a_i/b_i, d_i)$ is valid. By the same argument as in the proof of Theorem 4, the total cost of step (3) will be bounded by $O(\log^2 N)$ bit operations. By Lemma 1, step (3) sets $B[t] =$ false only if $\gcd(a + tb, N) > 1$. On the other hand, if $B[t]$ is still true after step (3) completes, then $a_i + tb_i \neq 0 \pmod{d_i}$ for all $1 \leq i \leq q$. By Theorem 2, there will exists at least one t between 0 and K with $B[t]$ still equal to true in step (4). By the fact that $a_i + tb_i \neq 0 \pmod{d_i}$ for $1 \leq i \leq q$, the gcd's g_i computed in step (5) will all satisfy $1 \leq g_i < d_i$ for $1 \leq i \leq q$. In particular, if $g_i > 1$ for some i, then this provides a nontrivial factorization of d_i in step (6). On the other hand, if $g_i = 1$ for $1 \leq i \leq q$, then we must have $\gcd(a + tb, N) = 1$. Finally, note that the complexity of step (4) is bounded by $O(\log^2 N)$ bit operations since $\prod_{1 \leq i \leq q} d_i = N$ and $|a_i|, |b_i| < d_i$ with $d_i > 1$ for $1 \leq i \leq q$.

Corollary 6 *Given a positive integer* N *and* k *pairs of numbers* $(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)$ *with* $\gcd(a_i, b_i) = 1$ *for* $1 \leq i \leq k$, *there exists a deterministic algorithm that computes, in succession for* $i = 1, 2, \ldots, k$, *the smallest nonnegative integer* c_i *that satisfies* $\gcd(a_i + c_i b_i, N) = 1$. *If* $|a_i|, |b_i| < N$ *for* $1 \leq i \leq k$, *then the running time of the algorithm is bounded by* $O(k \log^2 N + \log^3 N)$ *bit operations.*

Proof. Starting with the trivial factorization $[N]$ of length one, use algorithm `Conditioner` to compute c_i in succession for $i = 1, 2, \ldots, k$. Algorithm `Conditioner` requires repetition at most $k + \lfloor \log_2 N \rfloor$ times since the factorization $[N]$ can be refined (nontrivially) at most $\lfloor \log_2 N \rfloor$ times. ■

3 The modulo N extended gcd algorithm

Let N be a positive integer and

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix}$$

be an $1 \times n$ row vector. The modulo N extended gcd algorithm given in the introduction can be posed in terms of computing a certain $n \times n$ unimodular conditioning matrix

$$C = \begin{bmatrix} c_1 & & & \\ c_2 & 1 & & \\ c_3 & & 1 & \\ \vdots & & & \ddots \\ c_n & & & & 1 \end{bmatrix}.$$

Postmultiplying A by C has the effect of adding certain multiples of columns $2, 3, \ldots, n$ to column 1 of A. The conditioned matrix AC can be written as

$$AC = \begin{bmatrix} a'_n & a_2 & a_3 & \cdots & a_n \end{bmatrix}$$

with

$$a'_n = \sum_{i=1}^{n} c_i a_i \tag{3}$$

and should satisfy

$$\gcd(a'_n, N) = \gcd(a_1, a_2, \ldots, a_n, N). \tag{4}$$

Theorem 7 *There exists a deterministic algorithm that takes as input a positive integer* N *together with an integer vector* $[a_i]_{i=1}^n$, *and returns as output an integer vector* $[c_i]_{i=1}^n$ *which satisfies (4) with (3). Furthermore, the output vector will satisfy*

(*e*1) $c_1 = 1$.

(*e*2) *At most* $\lfloor \log_2 N \rfloor$ *of the* c_i*'s will be nonzero.*

(*e*3) $|c_i| \leq \lceil 2(\log_2 N)^{3/2} \rceil$ *for* $1 \leq i \leq n$.

If $|a_i| \leq N$ *for* $1 \leq i \leq N$, *then the running time of the algorithm is bounded by* $O(n \log^2 N + \log^3 N)$ *bit operations.*

Proof. The algorithm works by computing c_l in succession for $l = 1, 2, \ldots, k$. Let $c_1 = 1$ and define the intermediate values $a'_l = a_1 + c_2 a_2 + \cdots + c_l a_l \bmod N$. At the end of stage $l - 1$ and start of stage l, the quantities $c_1, c_2, \ldots, c_{l-1}$ have already been computed and satisfy

$$\gcd(a'_i, N) = \gcd(a_1, a_2, \ldots, a_i, N) \tag{5}$$

for $i = l - 1$. Note for the initial case $i = 0$ that condition (5) is trivially satisfied. The goal at stage l is to compute a suitable c_l such that (5) is satisfied for $i = l$. This is accomplished by choosing c_l to be the smallest nonnegative integer such that $\gcd(a'_{l-1}/g + c_l \cdot (a_l/g), N) = 1$ where $g = \gcd(a'_{l-1}, a_l)$. By Theorem 2, c_l will be bounded in magnitude by $\lceil 2(\log_2 N)^{2/3} \rceil$. This shows that condition (*e*3) is satisfied. We now show that condition (*e*2) is satisfied. Note that $\gcd(a'_i, N)$ is a divisor of $\gcd(a'_{i-1}, N)$ for $i = 1, 2, \ldots, k$ and if $\gcd(a_i, N) = \gcd(a_{i-1}, N)$ then c_i will have been chosen to be zero. Since N is $\lfloor 1 + \log_2 N \rfloor$ bits in length, there can be at most $\lfloor \log_2 N \rfloor$ distinct choices for i with $\gcd(a'_i, N)$ a proper divisor of $\gcd(a'_{i-1}, N)$. Here we are using the upper bound $\lfloor \log_2 N \rfloor$ for the number of prime divisors (not necessarily distinct) in the full factorization of N. This shows that at most $\lfloor \log_2 N \rfloor$ of the c_i's will be nonzero. The running time follows from Corollary 6. ■

4 Transforming matrices for the Smith normal form

Given an $n \times m$ rank m integer input matrix, we want to recover an $n \times n$ unimodular matrix U and an $m \times m$ unimodular matrix V such that $UAV = S$, where S is the Smith normal form of A. Some of the ideas we use in our algorithm have appeared previously in different contexts. For clarity, we prefer to present the new algorithm first and wait until Section 5 to indicate some interesting similarities with previous work. In Subsection 4.1 we give our algorithm for computing transforming matrices. In Subsection 4.2 we give some explicit examples of the new Smith normal form algorithm. Before continuing, we recall some definitions, define some notation and summarize some previous results.

The Smith normal form $S = UAV = \mathrm{diag}(s_1, s_2, \ldots, s_m)$ of A is obtained by applying unimodular row and column transformations to A. We denote the diagonal entries s_i by $s(A,i)$. The quantity $\det \mathcal{L}(A)$ — the determinant of the lattice of A — is equal to $s_1 s_2 \cdots s_m$ and is invariant under both unimodular row and column transformation. When A is square nonsingular, then $\det \mathcal{L}(A) = |\det A|$.

The Smith normal form can also be computed over the ring $\mathbf{Z}_d$ of integer modulo d. A matrix $\mathrm{diag}(s_1, s_2, \ldots, s_m) \in \mathbf{Z}_d^{n \times m}$ is in *Smith normal form* if $s_i | s_{i+1}$ for $i = 1, 2, \ldots, m-1$ and each s_i belongs to the set $N_d^* = \{x \bmod d : x \in \mathbf{Z}, 0 < x \leq d, x | d\}$. The set N_d^* is called a prescribed complete set of nonassociates of $\mathbf{Z}_d$ — specifying that the diagonal entries belong to N_d^* ensures uniqueness of the form. Over the ring $\mathbf{Z}$, the prescribed complete set of associates is simply the set of nonnegative integers.

It will be convenient to sometimes consider a matrix over $\mathbf{Z}$ to be over $\mathbf{Z}_d$ — simply reduce all entries modulo d. Conversely, any matrix over $\mathbf{Z}_d$ may be considered to be over $\mathbf{Z}$ — simply embed all entries from $\mathbf{Z}_d$ into $\mathbf{Z}$. If A is an integer matrix, we will write $s_d(A,i)$ to denote the i-th diagonal entry of the Smith normal form of A as computed over the ring $\mathbf{Z}_d$. The following result ensures that for certain values of d the Smith normal form of A as computed over $\mathbf{Z}_d$ will be the same as the Smith normal form of A over $\mathbf{Z}$.

Theorem 8 *If A is an $n \times m$ rank m integer matrix and d is a positive multiple of $2\det\mathcal{L}(A)$ then $s_d(A,i) = s(A,i)$ for $1 \leq i \leq m$.*

Proof. See, for example, [21, Theorem 12]. ∎

For $a, b \in \mathbf{Z}_d$, we write $\gcd_d(a,b)$ to denote the unique principal generator of the ideal $(a,b) \subseteq \mathbf{Z}_d$ which belongs to N_d^*. Note that $\gcd_d(a,b)$ can be computed as $\gcd(\bar{a}, \bar{b}, d) \bmod d$ where $\bar{a}$ and $\bar{b}$ are in $\mathbf{Z}$ with $a = \bar{a} \bmod d$ and $b = \bar{b} \bmod d$. For the case $a, b = 0$, we have $\gcd_d(0,0) = 0$. For complexity analysis we will sometimes count ring operations from $\mathbf{Z}_d$. Given two elements $a, b \in \mathbf{Z}_d$, a ring operation is one of: computing ab, $a - b$, $\gcd_d(a,b)$ or determining if $a|b$ and if so returning a c with $ac = b$. Each of these operations can be accomplished in $O(\log^2 d)$ bit operations using standard integer arithmetic.

We will need the following result.

Fact 2 (Hafner & McCurley [5]) *Let $A \in \mathbf{Z}_d^{n \times m}$ and let i and j be indices with $1 \leq i \leq n$ and $1 \leq j \leq m$. Working over the ring $\mathbf{Z}_d$, we can apply unimodular row transformations to transform A to a matrix B which satisfies $B_{ij} = \gcd_d(A_{i,j}, A_{i+1,j}, \ldots, A_{n,j})$ and $B_{kj} = 0$ for $i+1 \leq k \leq n$. The algorithm requires $O(nm)$ operations from $\mathbf{Z}_d$.*

4.1 The transforming matrix algorithm

Let A be an $n \times n$ nonsingular integral input matrix and $d = 2|\det A|$. Our algorithm for computing the Smith normal form S of A follows the mod d approach of many previous algorithms (see Hafner & McCurley [5]) and computes over the finite ring $\mathbf{Z}_d$ in order to avoid the problem of intermediate expression swell. Unlike most previous algorithms, we will also recover a unimodular U and V satisfying $UAV = S$. Since A is square nonsingular, it will be sufficient to recover a V and then compute U as $U \leftarrow SV^{-1}A^{-1}$. Since we are working over the ring $\mathbf{Z}_d$ we must take care that the V we produce will be unimodular over $\mathbf{Z}$ (and not just over $\mathbf{Z}_d$). Our approach is to compute a decomposition for V as the product of a unit lower triangular matrix C and unit upper triangular matrix R, namely

$$V = \underbrace{\begin{bmatrix} 1 & & & & \\ c_{21} & 1 & & & \\ c_{31} & c_{32} & 1 & & \\ \vdots & \vdots & & \ddots & \\ c_{n1} & c_{n2} & c_{n3} & \cdots & 1 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} 1 & r_{12} & r_{13} & \cdots & r_{1n} \\ & 1 & r_{23} & \cdots & r_{2n} \\ & & 1 & & r_{3n} \\ & & & \ddots & \vdots \\ & & & & 1 \end{bmatrix}}_{R} \quad (6)$$

Entries in column j of R will be bounded in magnitude by $s(A,j)$. Each column in C will be the solution to a particular instance of the modulo N extended gcd problem with $N = d$. The modulo N extended gcd algorithm of Corollary 6 will produce entries for C which admit the very small length bound of $O(\log \log d)$ bits; this bound on the size of entries in C leads to very pleasing bounds on the magnitudes of entries in $V \leftarrow CR$ and $U \leftarrow SR^{-1}C^{-1}A^{-1}$.

We now explain how to recover the matrices C and R. Initialize T to be a copy of the input matrix A. The first phase of the algorithm is to transform T using unimodular row and column operations over $\mathbf{Z}_d$ to an upper triangular matrix which has i-th diagonal entry $s_d(A,i)$. Recording column operations during this phase will produce the lower triangular matrix C of (6). The algorithm is recursive and can be understood by considering the first step which computes the entries $c_{21}, c_{31}, \ldots, c_{n1}$ of the first column of C. The goal at this first step is to compute a matrix

$$C_1 = \begin{bmatrix} 1 & & & & \\ c_{21} & 1 & & & \\ c_{31} & & 1 & & \\ \vdots & & & \ddots & \\ c_{n1} & & & & 1 \end{bmatrix}$$

which will comprise the entries in the first column of C. Note that postmultiplying T by C_1 will cause certain multiples of column $2, 3, \ldots, n$ to be added to column 1; the purpose of this is to ensure that the matrix TC_1 has $\gcd_d$ of all entries in the first column equal to the $\gcd_d$ of all entries in T. The entries c_{k1} are computed for $k = 2, 3, \ldots, m$ in succession. Let T' be the $n \times 2$ submatrix comprised of columns 1 and k of T. Compute a unimodular row transformation $B' \in \mathbf{Z}_d^{n \times 2}$ of T' such that B' has all entries below the first entry B'_{12} in column 2 zero. By Fact 2 this costs $O(n)$ ring operations from $\mathbf{Z}_d$. Because B' has rank 2 over $\mathbf{Z}$, entry B'_{1k} will be nonzero. Compute $g \leftarrow \gcd(B'_{11}, B'_{1k})$ and set $a \leftarrow B'_{11}/g$ and $b \leftarrow B'_{12}/g$. Using Algorithm `Conditioner` of Subsection 2.3 (with $N = d$) compute the smallest nonnegative integer t which satisfies $\gcd_d(a + tb) = \gcd_d(a,b)$. Set $c_{k1} \leftarrow t$ and add t times column k of T to column 1 of T. Note that adding t times column 2 of B' to column

1 of B' will ensure that the $\gcd_d$ of all entries in column 1 of B' is the $\gcd_d$ of all entries in columns 1 *and* 2 of B'. The key to our approach is the next statement: Since B' is a unimodular row transformation of columns 1 and k of T, adding t times column k of T to column 1 of T will ensure that the $\gcd_d$ of all entries in column 1 of T is now the $\gcd_d$ of all entries in columns 1 *and* k of T. After computing c_{k1} for $k = 2, 3, \ldots, m$ the work matrix T will have $\gcd_d$ of all entries in the first column be the $\gcd_d$ of all entries in T. Working over $\mathbf{Z}_d$, apply unimodular row transformations to the work matrix T so that

$$T = \left[\begin{array}{c|c} s_1 & * \\ \hline & T_1 \end{array}\right].$$

The conditioning of column 1 we performed earlier now ensures that s_1 will be the $\gcd_d$ of all entries of T, that is, $s_1 = s_d(T, 1)$. Column 2 of C is computed by applying the procedure just described to the $(n-1) \times (n-1)$ matrix T_1. At the end of the triangularization phase the work matrix T will be upper triangular with i-th diagonal entry equal to $s_i = s_d(A, i)$ for $1 \leq i \leq m$. By Theorem 8 we will have $s_i = s(A, i)$. Fact 2 and Corollary 6 bound the running time so far by $O(n^3)$ operations from $\mathbf{Z}_d$ plus an additional worst case cost of $O(\log^3 d)$ bit operations.

In phase two we zero out the offdiagonal entries in the now upper triangular work matrix T using unimodular row and column operations over $\mathbf{Z}$. Recording the column operations produces the matrix R row by row for row $i = n, n-1, \ldots, 1$ in succession. At stage $i = k$ we have

$$T = \left[\begin{array}{cccc|cccc} s_1 & * & \cdots & * & * & * & \cdots & * \\ & s_2 & \cdots & * & * & * & \cdots & * \\ & & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ & & & s_k & * & * & \cdots & * \\ \hline & & & & s_{k+1} & & & \\ & & & & & s_{k+2} & & \\ & & & & & & \ddots & \\ & & & & & & & s_n \end{array}\right]$$

and

$$R = \left[\begin{array}{ccc|ccc} 1 & & & & & \\ & 1 & & & & \\ & & \ddots & & & \\ & & 1 & & & \\ \hline & & & 1 & r_{k+1,k+1} & \cdots & r_{k+1,n} \\ & & & & 1 & \cdots & r_{k+2,n} \\ & & & & & \ddots & \vdots \\ & & & & & & 1 \end{array}\right]$$

By adding appropriate integer multiples of rows $k+1, k+2, \ldots, n$ of T to row k of T, ensure that entry T_{kj} satisfies $-s_j < T_{kj} \leq 0$ for $j = k+1, k+2, \ldots, n$. Now add $-T_{kj}$ times column k of T to column j for $j = k+1, k+2, \ldots, n$. Record these column operations in R. This zeroes out the offdiagonal entries in row k. Using integer row operations reduce all entries in the upper right hand $(k-1) \times (n-k-1)$ block of T modulo the diagonal entries in each column — this ensures that all entries in the work matrix T remain bounded by d. The computation of R is easily seen to be bounded by $O(n^3 \log^2 d)$ bit operations.

The two phase algorithm just described leads to the following.

Theorem 9 *There exists a deterministic algorithm that takes as input a nonsingular $A \in \mathbf{Z}^{n \times n}$ together with the quantity $d = 2|\det A|$, and returns as output the Smith normal form S of A together with a unimodular postmultiplier matrix V which satisfies $UAV = S$ where $U = SV^{-1}A^{-1}$ will also be unimodular. Entries in row j of V will be bounded in magnitude by $O(ns_j(\log d)^{3/2})$ where s_j is the j-th diagonal entry of S. If entries in A are bounded in magnitude by d, then the cost of the algorithm is $O(n^3 \log^2 d + \log^3 d)$ bit operations assuming standard integer arithmetic.*

The next result shows how to handle the case of rectangular input matrix with full column rank.

Theorem 10 *There exists a deterministic algorithm that takes as input a full column rank $A \in \mathbf{Z}^{n \times m}$, and returns as output the Smith normal form S of A together with unimodular transforming matrices U and V which satisfy $UAV = S$. Entries in U and V will be bounded in length by $O(m \log ||A||)$ bits. Furthermore, entries in row j of V will be bounded in magnitude by $O(ms_j(\log d)^{3/2})$ where s_j is the j-th diagonal entry of S and $d = s_1 s_2 \cdots s_m = O(m \log m||A||)$. The cost of the algorithm is $O(nm^3 \log^2 m||A|| + m^4 \log^3 m||A|| + m^3 \log^2 d)$ bit operations assuming standard integer arithmetic.*

Proof. First compute the Hermite normal form triangularization H of A together with a unimodular premultiplier matrix U_H which satisfies $U_H A = H$ and has entries bounded in length by $O(m \log m||A||)$ bits. This can be accomplished in $O(nm^3 \log^2 m||A|| + m^4 \log^3 m||A||)$ bit operations using the algorithm in [20]. Set $d = 2h_1 h_2 \cdots h_n$ where h_i is the i-th diagonal entry of H. Then $d = O(m \log m||A||)$. Let H_1 be the principal $m \times m$ submatrix of H so that H_1 is square nonsingular. Compute V and S using the algorithm of Theorem 9. It follows from the fact that H_1 is in Hermite normal form and from the bounds on the size of entries in V guaranteed by Theorem 9 that the matrix $U = SV^{-1}H_1^{-1}$ will have entries bounded in magnitude by $O(m \log m||A||)$. The matrix $U_1 \leftarrow SV^{-1}H_1^{\text{adj}}(1/d)$ can be computed in the allotted time using standard techniques. (For example, using a homomorphic imaging scheme with Chinese remaindering.) The premultiplier U is obtained from U_H by premultiplying the principal $m \times m$ block of U_H by U_1. ∎

4.2 Some explicit examples

The 9×9 square nonsingular input matrix

$$A = \begin{bmatrix} -7 & 8 & -21 & -29 & 7 & 0 & -10 & -3 & 1 \\ 3 & 6 & 0 & 0 & -7 & 30 & -12 & 5 & -1 \\ 13 & 15 & 6 & 13 & 4 & 10 & -5 & 14 & -10 \\ -1 & 6 & 12 & -10 & 3 & -7 & 10 & -13 & 14 \\ 9 & 10 & -3 & -11 & 11 & 11 & -6 & -7 & -14 \\ -4 & 6 & 3 & 5 & 10 & 2 & -24 & -6 & 12 \\ 5 & -8 & 18 & 0 & 13 & 13 & 6 & 15 & 8 \\ -11 & 11 & 18 & 9 & 0 & 17 & 7 & 4 & -7 \\ 2 & -1 & 33 & 4 & -9 & -3 & -19 & 5 & -12 \end{bmatrix}$$

has Smith normal form

$$S = \text{diag}(1, 1, 1, 1, 6, 30, 180, 6300, 44100).$$

Using the algorithm of Section 4 we can compute 9×9 unimodular transforming matrices U_{SEC4} and V_{SEC4} which satisfy $UAV = S$. In what follows, we write $[t]$ to indicate a integer with t decimal digits. Then

$$U_{\mathsf{SEC4}} = \begin{bmatrix} [6] & [6] & [6] & [6] & [6] & [5] & [3] & [6] & [5] \\ [6] & [6] & [6] & [6] & [6] & [6] & [3] & [6] & [5] \\ [5] & [5] & [5] & [5] & [5] & [5] & [3] & [6] & [5] \\ [6] & [6] & [6] & [6] & [6] & [6] & [4] & [7] & [6] \\ [6] & [6] & [6] & [6] & [6] & [6] & [4] & [6] & [5] \\ [6] & [6] & [6] & [5] & [5] & [5] & [3] & [6] & [5] \\ [6] & [6] & [6] & [6] & [6] & [6] & [4] & [7] & [6] \\ [6] & [6] & [6] & [6] & [6] & [6] & [4] & [7] & [6] \\ [7] & [7] & [7] & [6] & [6] & [6] & [4] & [7] & [6] \end{bmatrix}$$

and

$$V_{\mathsf{SEC4}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 3 & 20 & 76 & 3187 & 26944 \\ 1 & 1 & 0 & 0 & 4 & 28 & 119 & 4631 & 29411 \\ 3 & 0 & 1 & 0 & 10 & 84 & 311 & 13157 & 107658 \\ 0 & 1 & 4 & 1 & 6 & 114 & 433 & 19361 & 114992 \\ 1 & 1 & 1 & 0 & 6 & 52 & 207 & 8766 & 57501 \\ 1 & 1 & 1 & 0 & 6 & 53 & 208 & 8808 & 57966 \\ 1 & 1 & 1 & 9 & 15 & 150 & 738 & 40905 & 108351 \\ 1 & 1 & 0 & 1 & 5 & 39 & 181 & 8225 & 35525 \\ 1 & 1 & 0 & 0 & 4 & 29 & 122 & 4690 & 30184 \end{bmatrix}.$$

Entries in both U_{SEC4} and V_{SEC4} are bounded in length by 7 decimal digits. Note that for $1 \leq j \leq n$, entries in column j of V_{SEC4} are on the same order of magnitude as the j-th diagonal entry of S — just as guaranteed by Theorem 9. To get an idea of the total size required to write down V, let size(A) denote the sum of the lengths (number of decimal digits) of all entries in an integer matrix A. Then $\mathsf{size}(V_{\mathsf{SEC4}}) = 167$. For comparison, we give next the transforming matrices U_{MVR4} and V_{MVR4} returned by the current version of the `ismith` command in Maple V Release 4. The `ismith` command uses a variation which also computes transforming matrices of the modular determinant approach.

$$U_{\mathsf{MVR4}} = \begin{bmatrix} [0] & [0] & [0] & [1] & [0] & [0] & [0] & [0] & [1] \\ [0] & [0] & [0] & [1] & [0] & [0] & [0] & [0] & [1] \\ [0] & [0] & [0] & [2] & [0] & [1] & [1] & [0] & [1] \\ [0] & [4] & [0] & [7] & [0] & [5] & [5] & [3] & [6] \\ [12] & [16] & [12] & [19] & [11] & [17] & [17] & [15] & [18] \\ [12] & [17] & [13] & [20] & [11] & [18] & [18] & [16] & [19] \\ [19] & [27] & [22] & [30] & [23] & [28] & [28] & [26] & [29] \\ [29] & [37] & [32] & [39] & [32] & [38] & [38] & [36] & [39] \\ [30] & [38] & [32] & [40] & [33] & [39] & [39] & [37] & [40] \end{bmatrix}$$

and

$$V_{\mathsf{MVR4}} = \begin{bmatrix} [0] & [0] & [0] & [1] & [0] & [0] & [0] & [0] & [1] \\ [0] & [0] & [0] & [1] & [0] & [0] & [0] & [0] & [1] \\ [0] & [0] & [0] & [2] & [0] & [1] & [1] & [0] & [1] \\ [0] & [4] & [0] & [7] & [0] & [5] & [5] & [3] & [6] \\ [12] & [16] & [12] & [19] & [11] & [17] & [17] & [15] & [18] \\ [12] & [17] & [13] & [20] & [11] & [18] & [18] & [16] & [19] \\ [19] & [27] & [22] & [30] & [23] & [28] & [28] & [26] & [29] \\ [29] & [37] & [32] & [39] & [32] & [38] & [38] & [36] & [39] \\ [30] & [38] & [32] & [40] & [33] & [39] & [39] & [37] & [40] \end{bmatrix}$$

In this case, entries in U_{MVR4} and V_{MVR4} are bounded in length by 40 decimal digits and $\mathsf{size}(V_{\mathsf{MVR4}}) = 1489$. Thus, for this example the new algorithm returned a postmultiplier matrix that requires a factor of about nine times less space to write down.

More spectacular examples are easily generated. Consider the case of a 40×40 square nonsingular matrix

$$A = \begin{bmatrix} -3 & 3 & -5 & \cdots & -4 & -1 \\ 5 & -3 & 0 & \cdots & 3 & -1 \\ -4 & 0 & -2 & \cdots & -6 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 2 & -2 & 2 & \cdots & -1 & 0 \\ -1 & 1 & 0 & \cdots & -2 & 1 \end{bmatrix}$$

which has all entries bounded in magnitude by 9. The Smith normal form of A is comprised of 13 one's followed by $2, 2, 10, 10, 10, 6000, 36000$. The algorithm of Section 4 produces pre- and postmultipliers which have entries bounded in length by 14 decimal digits; the total size of the postmultiplier 691. Maple's `ismith` command produces a postmultiplier with entries bounded in length by 118 decimal digits and total size 12923; this is about 19 times as large as 691.

5 Conclusions

We have presented a solution for the modulo N extended gcd problem which returns an exceptionally small solution vector of multipliers. The gcd algorithm presented here has important application in computing canonical forms of integer matrices; in Section 4 we have presented a new algorithm for computing small transforming matrices for the Smith normal form of an integer matrix. We mention here some interesting similarities of our Smith normal form algorithm with previous work.

Our approach is to postmultiply the input matrix A by an $m \times m$ unit lower triangular "conditioning" matrix C such that the diagonal entries in the Hermite normal form triangularization of the conditioned matrix AC will be the same as those in the Smith normal form of A. Our Smith form algorithm computes very small entries for C using the modulo N extended gcd algorithm of Section 2. This idea of postmultiplying by a unit lower triangular matrix was first used by Kaltofen, Krishnamoorthy & Saunders [12, 13] in the context of computing Smith normal forms of polynomial matrices; there the matrix C was chosen randomly. Other

randomized algorithms using this preconditioning idea include Storjohann & Labahn [22, 23]. Villard [24] has given an algorithm — also for computing Smith normal forms of polynomial matrices — which computes *deterministically* the entries of C. More recently, Giesbrecht [3] has given a randomized Smith normal form algorithm for integer matrices that computes the columns of C by, in essence, using a Las Vegas probabilistic algorithm to obtain solutions to the modulo N extended gcd problem.

In the future, we plan to apply some of the ideas presented here to computational problems over other domains. For example, it should be possible to construct an algorithm which computes "good" (i.e. small degree) solutions for the extended Euclidean problem over the ring $\mathbf{F}[x]$ of univariate polynomials with coefficients from a field $\mathbf{F}$; this would be useful for computing canonical forms of polynomial matrices.

Finally, the modulo N extended gcd problem we have introduced here should be analysed along the lines of [15] where the complexity of finding an optimal solution to the extended gcd problem with respect to the L_∞ and L_0 norm is studied.

References

[1] COPPERSMITH, D., AND WINOGRAD, S. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation 9* (1990), 251–280.

[2] FORD, D., AND HAVAS, G. A new algorithm and refined bounds for extended gcd computation. Tech. Rep. 354, The University of Queensland, 1995. Submitted.

[3] GIESBRECHT, M. Fast computation of the Smith normal form of an integer matrix. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '95* (1995), A. H. M. Levelt, Ed., pp. 110–118.

[4] GIESBRECHT, M. Probabalistic computation of the Smith normal form of a sparse integer matrix. In *Algorithmic Number Theory: Second International Symposium* (1996), H. Cohen, Ed., pp. 175–188. Proceedings to appear in Springer's Lecture Notes in Computer Science.

[5] HAFNER, J. L., AND MCCURLEY, K. S. Asymptotically fast triangularization of matrices over rings. *SIAM Journal of Computing 20*, 6 (Dec. 1991), 1068–1083.

[6] HAVAS, G., AND MAJEWSKI, B. S. Diagonalization of integer matrices. To appear in *Journal of Symbolic Computation*.

[7] HAVAS, G., AND MAJEWSKI, B. S. Hermite normal form computation for integer matrices. *Congressus Numerantium 105* (1994), 87–96.

[8] HAVAS, G., AND MAJEWSKI, B. S. A hard problem that is almost always easy. *Algorithms and Computation, Lecture Notes in Computer Sciece 1004* (1995), 216—223.

[9] HAVAS, G., MAJEWSKI, B. S., AND MATTHEWS, K. R. Extended gcd algorithms. Tech. Rep. 302, The University of Queensland, 1995. Submitted.

[10] HOWELL, J. A. Spans in the module $(\mathbf{Z}_m)^s$. *Linear and Multilinear Algebra 19* (1986), 67—77.

[11] ILIOPOULOS, C. S. Worst-case complexity bounds on algorithms for computing the canonical structure of finite abelian groups and the Hermite and Smith normal forms of an integer matrix. *SIAM Journal of Computing 18*, 4 (Aug. 1989), 658–669.

[12] KALTOFEN, E., KRISHNAMOORTHY, M. S., AND SAUNDERS, B. D. Fast parallel computation of Hermite and Smith forms of polynomial matrices. *SIAM Journal of Algebraic and Discrete Methods 8* (1987), 683–690.

[13] KALTOFEN, E., KRISHNAMOORTHY, M. S., AND SAUNDERS, B. D. Parallel algorithms for matrix normal forms. *Linear Algebra and its Applications 136* (1990), 189–208.

[14] KANOLD, H.-J. Über eine zahlentheoretische Function von E. Jacobsthal. *Abh. Braunschwieg. Wiss. Gesellsch. 25* (1975), 7–10.

[15] MAJEWSKI, B. S., AND HAVAS, G. The complexity of greatest common divisor computations. *Algorithmic Number Theory, Lecture Notes in Computer Science 877* (1994), 184—193.

[16] MAJEWSKI, B. S., AND HAVAS, G. A solution to the extended gcd problem. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '95* (1995), A. H. M. Levelt, Ed., pp. 248—253.

[17] SCHÖNHAGE, A., AND STRASSEN, V. Schnelle Multiplikation grosser Zahlen. *Computing 7* (1971), 281–292.

[18] SMITH, H. J. S. On systems of linear indeterminate equations and congruences. *Phil. Trans. Roy. Soc. London 151* (1861), 293–326.

[19] STORJOHANN, A. Computing Hermite and Smith normal forms of triangular integer matrices. Tech. Rep. 256, Departement Informatik, ETH Zürich, Dec. 1996.

[20] STORJOHANN, A. A fast+practical+deterministic algorithm for triangularizing integer matrices. Tech. Rep. 255, Departement Informatik, ETH Zürich, Dec. 1996.

[21] STORJOHANN, A. Near optimal algorithms for computing Smith normal forms of integer matrices. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '96* (1996), Y. N. Lakshman, Ed., ACM Press, pp. 267–274.

[22] STORJOHANN, A., AND LABAHN, G. Preconditioning of rectangular polynomial matrices for efficient Hermite normal form computation. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '95* (1995), A. H. M. Levelt, Ed., ACM Press, pp. 119–125.

[23] STORJOHANN, A., AND LABAHN, G. A fast Las Vegas algorithm for computing the Smith normal form of a polynomial matrix. *Linear Algebra and its Applications 253* (1997), 155—173.

[24] VILLARD, G. Generalized subresultants for computing the Smith normal form of polynomial matrices. *Journal of Symbolic Computation 20*, 3 (95), 269—286.