# Hermite normal form computation for integer matrices

George Havas[*] and Bohdan S. Majewski[†]
Key Centre for Software Technology
Department of Computer Science
The University of Queensland
Queensland 4072, Australia

**Abstract**

We consider algorithms for computing the Hermite normal form of integer matrices. Various different strategies have been proposed, primarily trying to avoid the major obstacle that occurs in such computations — explosive growth in size of intermediate entries. We analyze some methods for computing the Hermite normal form and we show the intractability of associated problems. We investigate in detail a method based on an algorithm due to Blankinship and show how improved performance is achieved.

## 1 Introduction

Integer matrices $A$ and $B$ are row equivalent if there exists a unimodular matrix $P$ such that $A = PB$. Matrix $P$ corresponds to a sequence of elementary row operations: negating a row; adding an integer multiple of one row to another; or interchanging two rows. It follows from a result of Hermite [Her51] that for any integer matrix $B$ there exists a unique upper triangular matrix $H$, which satisfies the following conditions.
• Let $r$ be the rank of $B$. Then the first $r$ rows of $H$ are nonzero.
• For $1 \leq i \leq r$ let $H[i, j_i]$ be the first nonzero entry in row $i$. Then $j_1 < j_2 < \ldots < j_r$.
• $H[i, j_i] > 0$, for $1 \leq i \leq r$.
• For $1 \leq k < i \leq r$, $H[i, j_i] > H[k, j_i] \geq 0$.

This matrix is called the row Hermite normal form (HNF) of the given matrix $B$ and has many important applications. There is a "standard" algorithm for computing the HNF, based on Gaussian elimination. Unfortunately the standard algorithm suffers from serious practical difficulties. Modular methods can be used, but the complexity bounds have relatively high degree when compared to integer based methods. That is why we study integer methods here.

[*]**E-mail**: havas@cs.uq.oz.au; partially supported by the Australian Research Council
[†]**E-mail**: bohdan@cs.uq.oz.au

Many of the problems which we address here are very similar to problems which arise in the related task of computing another canonical form of integer matrices, the Smith normal form (SNF). Computation of that form is studied in detail in [HHR93], which provides background material also relevant to HNF calculation. It presents detailed justifications for the use of integer as against modular methods, and includes a comprehensive bibliography plus a number of examples.

In this paper we indicate the difficulties that need to be faced in pursuing an efficient method for computing the Hermite normal form of an integer matrix. We present and analyze a method which gives more efficient solutions.

We use the following notation. For a $m \times n$ integer matrix $B$ we denote its $i$-th row by $\mathbf{b}_{i*}$ and its $j$-th column by $\mathbf{b}_{*j}$. The absolute value of $x$ is denoted by $|x|$, while $\det(B)$ stands for the determinant of $B$. We denote by $||B||$ the maximum absolute value of any entry in $B$. Matrix $B$ may be alternatively written as

$$B = [\mathbf{b}_{*1}, \ldots, \mathbf{b}_{*n}] = \left[ \begin{array}{c} \mathbf{b}_{1*} \\ \vdots \\ \mathbf{b}_{m*} \end{array} \right].$$

## 2 Existing methods

Our primary goal in Hermite normal form computations is to minimize the size of the intermediate entry with maximum magnitude. In this section we discuss some existing methods and describe how they deal with that problem.

Kannan and Bachem [KB79] compute the Hermite normal form of an integer matrix $B$ by putting the principal minors of $B$ into HNF. They prove that such an approach results in polynomial time algorithms. Chou and Collins [CC82] improve the bounds given in [KB79] by modifying the procedure for normalizing entries above the main diagonal. This approach has merits in that it provides a method whose complexity can be formally analyzed. However it seems hard to modify in a way which keeps the basic method but improves the performance. The main reason is the inherently binary character of the method, in the sense that each of its row operations involves at most two rows.

In general such a binary approach suffers greatly from a lack of broader perspective. Consider the $i$-th step of the algorithm. In this step, the $i$-th row needs its leading entries forced to 0. For $j < i$, the $j$-th entry, once made divisible by $b_{jj}$, is changed to 0 by adding row $j$ multiplied by $-b_{ij}/\gcd(b_{jj}, b_{ij})$ to row $i$. Most of the time $\gcd(b_{jj}, b_{ij})$ is a small constant, 1 being very common for large matrices. Hence the entries in the $i$-th row may essentially be squared each time an entry is forced to zero. More precisely we have:

**Theorem 1** *[KB79, Lemma 2] At any stage of the execution of the algorithm the largest entry in $B$ does not exceed $2^{2n} n^{20n^3} ||B^1||^{12n^3}$.*

Here $B^1$ denotes the original matrix, which has size $n \times n$, and $B$ is the working matrix. Thus, although the algorithm is polynomial, the upper bound on the length of numbers it may operate on is in excess of $20n^3$ digits.

An alternative way to compute the Hermite normal form is to compute it column by column [Bod56, Bra71, Fru76, Hu69, PB74], which necessitates forcing all entries in a column below the main diagonal to 0. Such a technique allows greater freedom, as more than two rows at a time can participate in calculations. Unfortunately, as often reported in the literature, many methods that use this approach perform quite badly.

In a study on SNF calculation in [HHR93], there are examples where, after a small number of steps, naive methods create intermediate entries hundreds of digits long. On the other hand a number of heuristics are given in [HHR93] that perform very well in practice. A good example is provided by their $26 \times 27$ integer matrix called $R_1$. The entry with largest magnitude in the initial matrix $R_1$ is $-3$, and 46% of all entries are 0. In this case, pivoting on the maximum entry generates an entry with 1626 decimal digits in an incomplete calculation during which only the first 12 columns are cleared below the diagonal. Pivoting on the first nonzero element gives a largest entry with 262 decimal digits in completely computing the HNF. The algorithms of Kannan-Bachem and Chou-Collins both lead to 12 decimal digit numbers, which needs multiple precision on 32 bit machines. Heuristic approaches, which take into account issues considered in this paper, complete the process with 7 decimal digit largest entries. In fact, excluding a final column adjustment step (which converts the matrix from a row echelon form to HNF), the largest entry is the same as the largest entry in the final output, 5 decimal digits.

In the following section we look at the difficulties and give reasons for the bad performance of naive methods. Next we explain why heuristic methods like those presented in [HHR93] perform so much better.

## 3 Entry explosion

We distinguish two different techniques. The first one, which is simpler, computes the gcd of a number of elements (expressed as their linear combination) explicitly ([Bod56, Bra71, Hu69]). In other words, in the $i$-th step we ask for an integer vector $\mathbf{x}$, such that

$$\mathbf{x} \cdot \mathbf{b}_{*i} = \gcd(b_{ii}, \ldots, b_{ni}).$$

Once this is achieved we use the $i$-th row to force the remaining entries to 0. (The leading entry of the $i$-th row divides any other entry in the $i$-th column, hence forcing any entry to 0 is a straightforward operation.)

The simplicity of this approach relies on the fact that at each step we are only concerned with gcd computations for $n$ numbers, a problem which is studied in detail in [MH94]. Unfortunately, the method has significant disadvantages. Let us analyze the method by looking at the operations performed for the first column. Firstly we have the issue of finding a short vector $\mathbf{x}$.

**Theorem 2** *[MH94] For a given vector of $n$ positive integers $\mathbf{a} = [a_1, \ldots, a_n]$ the task of finding the shortest, with respect to either the $L_0$ metric or the $L_\infty$ norm, vector $\mathbf{x}$ which solves $\mathbf{x} \cdot \mathbf{a} = \sum_{i=1}^{n} x_i a_i = \gcd(a_1, \ldots, a_n)$ is NP-hard.*

Furthermore, small multipliers will not necessarily help us a lot. In all likelihood the gcd of the entries in the first column, for large $n$, will be 1. Then forcing $b_{i1}$ to 0 will require subtracting row 1 multiplied by $b_{i1}$ from row $i$. In other words, the unimodular matrix $P$, such that $PB$ has has only one nonzero entry in the first column, will have the following form:

$$P = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_m \\ -b_{21}x_1 & 1-b_{21}x_2 & -b_{21}x_3 & \dots & -b_{21}x_m \\ -b_{31}x_1 & -b_{31}x_2 & 1-b_{31}x_3 & \dots & -b_{31}x_m \\ \vdots & & & \ddots & \\ -b_{n1}x_1 & -b_{n1}x_2 & -b_{n1}x_3 & \dots & 1-b_{n1}x_m \end{bmatrix} \tag{1}$$

To compute the HNF of $B$ we need to calculate $k = \min(m, n)$ such unimodular matrices. It follows that even if we could find short vectors $\mathbf{x}$ (say, meaning $|x_i| \le c$, for some constant $c$) the best bound on the maximum entry in $B$, before the process of column normalization takes place, is $O(||B||^{2^k})$. This gives the algorithm no better than a clearly exponential upper bound on time complexity. (Frumkin [Fru76] gives the bound of $O(||B||^{3^k})$; this occurs if $x_i = O(||B||)$, which is often the case for existing gcd algorithms.)

Alternatively, we could ask for a vector $\mathbf{x}$ which minimizes the new entries in $B$. This however is not an easy task. The expression for the new value of $b_{ij}$ is

$$b_{ij} \leftarrow \sum_{k=1}^{m} x_k \frac{\det\left(\begin{bmatrix} b_{k1} & b_{kj} \\ b_{i1} & b_{ij} \end{bmatrix}\right)}{\gcd(b_{11}, \dots, b_{m1})}. \tag{2}$$

Consider now a single new row of $B$. Denote by $\mathbf{d}_{ij}^{\mathrm{T}} = [d_{ij1}, \dots, d_{ijn}]$ the vector of coefficients associated with the $x_k$'s in (2), $d_{ijk} = (b_{k1}b_{ij} - b_{kj}b_{i1})/\gcd(b_{11}, \dots, b_{m1})$. Each $b_{ij}$ can be expressed as $\mathbf{d}_{ij} \cdot \mathbf{x}$. Hence the $i$-th row of $B$, $\mathbf{b}_{i*}$, can be computed as

$$\mathbf{b}_{i*} = T_i \mathbf{x}^T = \sum_{i=1}^{m} \mathbf{t}_{*i} x_i \tag{3}$$

where $T_i = [t_{pq}]$, with $t_{pq} = d_{ipq}$. The last equation presents a rather unpleasant consequence. The problem of minimizing the maximum entry in $\mathbf{b}_{i*}$, where $\mathbf{b}_{i*}$ is expressed as a linear combination of $n$ vectors, is NP-hard [vEB81].

A second technique we can employ is to find a method of computing the transforming, unimodular matrix $P$ in such a way that the entries in each row are small. (In this method we compute the **entire** matrix $P$ in one go, instead of computing its first vector which, together with matrix $B$, determines the remaining vectors of $P$.) As a starting point we use the algorithm of Blankinship [Bla63]. Originally, the purpose of the algorithm was to express the gcd of $n \ge 2$ numbers as their linear combination. However the algorithm actually produces a unimodular matrix $P$ such that, for a vector of $n$ integers $\mathbf{a} = [a_1, \dots, a_n]^T$, we have $\sum_{j=1}^{n} p_{1j}a_j = \gcd(a_1, \dots, a_n)$ and $\sum_{j=1}^{n} p_{ij}a_j = 0$, for $2 \le j \le n$. Thus, such algorithms can be utilized to handle a matrix $B$ column by column.

For completeness we give an outline of Blankinship's method here. In the first step, set $P$ equal to $I_n$, an $n \times n$ identity matrix. Select the smallest element in $\mathbf{a}$, say

$a_i$. Select any other nonzero element in $\mathbf{a}$, say $a_j$. Compute $a_j = qa_i + r$ and replace $a_j$ by $r$. Apply this same operation to matrix $P$ by subtracting $q$ times row $i$ from row $j$. Repeat this process until only one nonzero element in $\mathbf{a}$ is left.

Suppose that the method of selecting a nonzero element of $\mathbf{a}$ is to choose the smallest $j \neq i$ for which $a_j \neq 0$. Let $a_1 = \min(a_1, \ldots, a_n)$ and, for simplicity, assume that $\gcd(a_1, a_2) = \gcd(a_1, \ldots, a_n)$ (an event quite likely to occur). Blankinship's algorithm then starts by computing $p_{11}a_1 + p_{12}a_2 = \gcd(a_1, a_2)$. Next, rows 3 to $n$ will be modified by subtracting $\mathbf{p}_{1*} \times a_k / \gcd(a_1, a_2)$, for $k = 3, \ldots, a_n$. As a result, matrix $P$ may have entries as large as $\max(a_i)^2$, which we see witnessed later in our example and which is clearly undesirable. Thus the Blankinship method, without some modification, is unattractive. In addition, determining an optimum matrix $P$, with respect to the $L_\infty$ norm, is an NP-hard task, as shown by the following theorem.

**Theorem 3** *The problem of minimizing the vectors, with respect to the maximum norm, of the final matrix given by a Blankinship-type algorithm is NP-hard.*

**Proof.** We give a transformation between a known NP-hard problem and our task. The BOUNDED HOMOGENOUS LINEAR EQUATION (BHLE) problem [vEB81] asks if, for a vector of $n$ integers $[a_1, \ldots, a_n]$, there exists a nontrivial solution to the equation $\sum_{i=1}^{n} x_i a_i = 0$ such that each $|x_i|$ is bounded above by some predefined constant $K$. This problem is NP-complete. The transformation is straightforward. By applying Blankinship's algorithm to the vector $\mathbf{a}$ we obtain $n-1$ bounded homogeneous linear equations. The ability to carry out polynomially bounded computations during Blankinship's method in such a way that the rows of the final matrix are optimal with respect to the maximum norm would imply the ability to solve the BHLE problem. $\square$

Obtaining small entries in $P$ is vital, even if computing 'the best' possible matrix $P$ is hard. If we could design a method that bounds the size of $|p_{ij}|$ by some constant, independent of the numbers in $\mathbf{a}$, we would immediately have a method that guarantees polynomial time complexity for Hermite norm computations based on it. This however is impossible, as indicated by the next lemma.

**Lemma 4** *There is a vector $\mathbf{a} = [a_1, a_2, \ldots, a_n]^T$, such that the absolute value of the largest entry in any matrix $P = [p_{ij}]$, such that $P\mathbf{a} = [\gcd(a_1, \ldots, a_n), 0, \ldots, 0]^T$, is equal to the largest entry in $\mathbf{a}$.*

**Proof**. Simply consider the vector $[1, a_2, \ldots, a_n]^T$, where $a_2 = a_3 = \ldots = a_n > a_1$. One solution is given by the matrix

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ -a_2 & 1 & 0 & 0 & \ldots & 0 \\ 0 & -1 & 1 & 0 & \ldots & 0 \\ 0 & 0 & -1 & 1 & \ldots & 0 \\ \vdots & & & & \ddots & \\ 0 & 0 & 0 & 0 & \ldots & 1 \end{bmatrix}$$

In finding any solution, whatever order is chosen for Blankinship type operations, the last remaining entry with value $a_2$ can only be removed by subtracting $a_2$ times 1, necessitating an entry in the solution matrix whose absolute value is $a_2$. Thus the maximum entry in $P$ must remain as large as $\max(a_i)$. $\square$

Thus we know that calculating a unimodular matrix $P$ with small entries is intractable, unless P = NP. We also know that any algorithm that guarantees $||P||$ to be no worse than $\max(a_i)$ is, in a sense, optimal. In the following section we provide two ways of significantly improving Blankinship's algorithm and provide some analysis of them.

(Note that the proof of Lemma 4 relies on the fact that there are only 2 distinct numbers in the vector. An interesting question to ask is: what is the worst case for a vector with $n$ numbers of which a specified minimum are distinct?)

## 4 Improvements to the algorithm of Blankinship

Blankinship's method, like the algorithms of Kannan and Bachem [KB79] and Chou and Collins [CC82], suffers from narrowness of its horizon. The selected entry in **a** is used to reduce only one other entry. If it happens that we keep on reducing two entries that have the same gcd as the whole column, then Blankinship's method ends up using only two rows and generating only two multipliers. Intuitively, using more than two rows, possibly all $n$ rows, should give us some benefits. One simple modification that results in very good overall performance is the following. The selected element is used to reduce **all** other elements of **a**. Hence one row of $P$ is subtracted (possibly 0 times for some rows) from all other rows of $P$. In this section we give examples and analyze the impact of such a strategy for the relatively simple case of just 3 numbers.

**Example**. Consider the vector $[f_n, f_{n+1}, f_{n+2} - 1]^T$, where $f_i$ is the $i$-th Fibonacci number. The matrix $P$ given by Blankinship's algorithm for this vector is (after swapping the first two rows to make the first row the gcd constructing row)

$$\begin{bmatrix} -f_{n-1} & f_{n-2} & 0 \\ f_{n+1} & -f_n & 0 \\ -(f_{n+2}-1)f_{n-1} & (f_{n+2}-1)f_{n-2} & 1 \end{bmatrix}$$

Observe that two entries in the third row are quite large, approximately squaring initial vector entries. Now suppose we perform the same basic sequence of operations, except that we subtract the operator row from both other rows. (During Blankinship's odd numbered steps we subtract the first row from the second **and third**, and during the even numbered steps, which are unchanged for this vector, we subtract the second row from the first.) As a result we obtain

$$\begin{bmatrix} -f_{n-1} & f_{n-2} & 0 \\ f_{n+1} & -f_n & 0 \\ -(f_{n-1}+1) & f_{n-2}-1 & 1 \end{bmatrix}$$

Notice the big reduction in the sizes of two entries in the last row, which are now linear in the initial vector entries. Even better performance is obtained if we use the best remainder strategy ([HHR93]). Then we obtain

$$\begin{bmatrix} 1 & 1 & -1 \\ f_{n-4}-5 & f_{n-4}+3 & -f_{n-4} \\ f_{n-2}-2 & f_{n-2}+1 & -f_{n-2} \end{bmatrix}$$

Thus the advantage of using the operator row on all rows instead of just one is quite pronounced. $\square$

Now we analyze the following modification of Blankinship's method, which is based on the method used in the second case above. For a vector of 3 integers $[a_1, a_2, a_3]^T$, we start by permuting the vector in such a way that $a_2 \geq a_3$. Next we set up matrix $P$, with $P = I_3$ initially. For the rest of this section we use superscripts to indicate algorithm steps. Thus, $p_{ij}^k$ denotes the value of $p_{ij}$ after the $k$-th step of the algorithm.

In every odd step, subtract $q_2^{2k-1}$ times $a_1$ from $a_2$ and $q_3^{2k-1}$ times $a_1$ from $a_3$, where $q_2^{2k-1} = \lfloor a_2/a_1 \rfloor$ and $q_3^{2k-1} = \lfloor a_3/a_1 \rfloor$. Apply the same operations to the rows of $P$. In every even step, execute the same operations, but this time subtracting $a_2$ from $a_1$ and $a_3$, $q_1^{2k}$ and $q_3^{2k}$ times, respectively, and again do the same to $P$. The algorithm stops when either $a_1$ or $a_2$ becomes 0 and the other is $\gcd(a_1, a_2)$.

**Lemma 5** *Throughout the modified algorithm the following holds*

$$\left.\begin{array}{r} |p_{31}| \leq \max(|p_{11}|, |p_{21}|) \\ |p_{32}| \leq \max(|p_{12}|, |p_{22}|) \end{array}\right\} \quad \text{always}$$

$$\left.\begin{array}{r} |p_{31}| \leq |p_{21}| \\ |p_{32}| \leq |p_{22}| \end{array}\right\} \quad \text{in the odd steps}$$

$$\left.\begin{array}{r} |p_{31}| \leq |p_{11}| \\ |p_{32}| \leq |p_{12}| \end{array}\right\} \quad \text{in the even steps}$$

**Proof.** (Informally, the proof is simple. The sizes of $p_{11}$, $p_{12}$, $p_{21}$ and $p_{22}$ constantly grow, maintaining the same sign throughout execution of the algorithm. However, $p_{31}$ and $p_{32}$ may vary in sign, but never have enough time to outgrow the above four entries. The worst case occurs if the third row is modified only half the time, always by the same row, so that it does not change sign. Notice that the third row cannot be modified more often than the second row. Consequently, on average, the third row cannot 'keep up' with the other two rows, and during some iterations it will be modified only by one of rows 1 and 2. Indeed, the size of the entries in the third row will often be much smaller than the entries in rows 1 and 2.)

Initially we have $p_{11}^0 = 1$, $p_{12}^0 = 0$, $p_{21}^0 = 0$, $p_{22}^0 = 1$, $p_{31}^0 = p_{32}^0 = 0$. Observe that in every odd step $q_2^{2k-1} \geq q_3^{2k-1}$ and in every even step $q_1^{2k} \geq q_3^{2k}$. Furthermore, observe that throughout the algorithm $p_{11} > 0$, $p_{12} \leq 0$, $p_{21} \leq 0$ and $p_{22} > 0$.

After the first step, as $q_2^1 \geq q_3^1$, we have $|p_{31}^1| \leq |p_{21}^1|$, both negative. After the second step $p_{31}^2 \leftarrow p_{31}^1 - q_3^2 p_{21}^1 = p_{31}^1 + q_3^2 |p_{21}^1|$ and $p_{11}^2 \leftarrow 1 + q_1^2 |p_{21}^1|$ and therefore $|p_{31}^2| \leq |p_{11}^2|$. Again, trivially, $|p_{32}^2| \leq |p_{12}^2|$. Thus we have established that, for $k = 1$, the following inequalities hold:

$$\begin{array}{lcl} |p_{31}^{2k}| \leq |p_{11}^{2k}| & \qquad & |p_{31}^{2k-1}| \leq |p_{21}^{2k-1}| \\ |p_{32}^{2k}| \leq |p_{12}^{2k}| & \qquad & |p_{32}^{2k-1}| \leq |p_{22}^{2k-1}| \end{array}$$

Now assume that these inequalities hold for some $k \geq 1$. In the next step, $2k+1$, the following modifications occur:

$$\begin{array}{lcl} p_{31}^{2k+1} & \leftarrow & p_{31}^{2k} - q_3^{2k+1} p_{11}^{2k} \\ p_{32}^{2k+1} & \leftarrow & p_{32}^{2k} - q_3^{2k+1} p_{12}^{2k} \\ p_{21}^{2k+1} & \leftarrow & p_{21}^{2k} - q_2^{2k+1} p_{11}^{2k} \\ p_{22}^{2k+1} & \leftarrow & p_{22}^{2k} - q_2^{2k+1} p_{12}^{2k} \end{array}$$

In order to establish $|p_{31}^{2k+1}| \leq |p_{21}^{2k+1}|$ we must verify that $|p_{31}^{2k} - q_3^{2k+1} p_{11}^{2k}| \leq |p_{21}^{2k} - q_2^{2k+1} p_{11}^{2k}|$. As $p_{21}^{2k}$ is negative, the last inequality is violated if either $p_{31}^{2k} - q_3^{2k+1} p_{11}^{2k} > |p_{21}^{2k}| + q_2^{2k+1} p_{11}^{2k}$ (for $p_{31}^{2k}$ positive) or if $|p_{31}^{2k}| + q_3^{2k+1} p_{11}^{2k} > |p_{21}^{2k}| + q_2^{2k+1} p_{11}^{2k}$ (for $p_{31}^{2k}$ negative). In the former case we would have $p_{31}^{2k} > |p_{21}^{2k}| + p_{11}^{2k}(q_3^{2k+1} + q_2^{2k+1})$, which is impossible, as $|p_{31}^{2k}| \leq |p_{11}^{2k}|$ and $q_3^{2k+1} + q_2^{2k+1} \geq 1$. In the latter case, to disprove our claim, we observe that

$$\begin{array}{rcl} |p_{31}^{2k+1}| > |p_{21}^{2k+1}| & \iff & |p_{31}^{2k}| + q_3^{2k+1} p_{11}^{2k} > |p_{21}^{2k}| + q_2^{2k+1} p_{11}^{2k} \\ & \iff & \left| p_{31}^{2k-1} + q_3^{2k} |p_{21}^{2k-1}| \right| + q_3^{2k+1} p_{11}^{2k} > |p_{21}^{2k-1}| + q_2^{2k+1} p_{11}^{2k} \\ & \iff & q_3^{2k} |p_{21}^{2k-1}| - |p_{31}^{2k-1}| + q_3^{2k+1} p_{11}^{2k} > |p_{21}^{2k-1}| + q_2^{2k+1} p_{11}^{2k} \end{array}$$

Notice that $q_3^{2k+1} p_{11}^{2k} \leq q_2^{2k+1} p_{11}^{2k}$, hence in order to fulfil the last inequality we need $q_3^{2k} |p_{21}^{2k-1}| - |p_{31}^{2k-1}| > |p_{21}^{2k-1}|$, which is equivalent to $|p_{31}^{2k-1}| < |p_{21}^{2k-1}|(q_3^{2k} - 1)$. However, this case was considered under the assumption that $p_{31}^{2k}$ is negative, namely, $p_{31}^{2k-1} + q_3^{2k}|p_{21}^{2k-1}| < 0 \iff |p_{31}^{2k-1}| > q_3^{2k}|p_{21}^{2k-1}|$, so we have a contradiction. Thus we proved that after the $(2k+1)$-st step $|p_{31}^{2k+1}| \leq |p_{21}^{2k+1}|$. Next we check if $|p_{32}^{2k+1}| \leq |p_{22}^{2k+1}|$, which amounts to $\left| p_{32}^{2k} + q_3^{2k} |p_{12}^{2k}| \right| \leq p_{22}^{2k} + q_2^{2k+1} |p_{12}^{2k}|$. If $p_{32}^{2k}$ is negative we can see that the inequality is true by the same argument as for the negative case for $p_{31}^{2k+1}$. If $p_{32}^{2k}$ is positive we require $p_{32}^{2k} \leq p_{22}^{2k}$. However $p_{32}^{2k} = p_{32}^{2k-1} - q_3^{2k} p_{22}^{2k-1}$ and $|p_{32}^{2k-1}| \leq p_{22}^{2k-1}$, so it follows that if $p_{32}^{2k} > 0$ then $p_{32}^{2k} \leq p_{22}^{2k}$. This finishes our proof for the $(2k+1)$-st step.

To complete the proof, we need to check the relations that hold in the $(2k+2)$-nd step. The analysis is similar, and hence the claimed inequalities are true. □

**Theorem 6** *There is a modification of Blankinship's algorithm for three numbers whose performance is close to the optimal bound, namely no entry in the final matrix exceeds* $(5/4)\max(a_1, a_2, a_3)$.

**Proof.** By Lemma 5, if $\gcd(a_1, a_2) = g_2$ divides $a_3$ evenly, the previous modification suffices. If $(a_3 \bmod g_2) = r_3 \neq 0$, we may split the algorithm into two phases: the first runs as described above; and the second, where the gcd of $g_2$ and $r_3$ is computed. In this second phase we apply Blankinship's original algorithm (which, for two numbers, is essentially a standard extended gcd calculation) to the vector $[g_2, 0, r_3]^T$. For convenience we assume $\gcd(a_1, a_2, a_3) = 1$. The final matrix $P$ is then equal to

$$P = \begin{bmatrix} y_1 & 0 & y_2 \\ 0 & 1 & 0 \\ -r_3 & 0 & g_2 \end{bmatrix} \times \begin{bmatrix} x_1 & x_2 & 0 \\ -a_2/g_2 & a_1/g_2 & 0 \\ p_{31} & p_{32} & 1 \end{bmatrix}$$

Here $y_1 g_2 + y_2 r_3 = \gcd(a_1, a_2, a_3) = 1$. It is possible (cf. [Lev56]) to execute both phases in such a way that $|x_1| \leq a_2/(2g_2)$, $|x_2| \leq a_1/(2g_2)$, $|r_3| \leq g_2/2$, $|y_1| \leq r_3/2$, $|y_2| \leq g_2/2$, $|p_{31}| \leq a_2/g_2$ and $|p_{32}| \leq a_1/g_2$ (the last two inequalities follow from Lemma 5). It then suffices to verify that

$$\begin{aligned} |y_1 x_1 + y_2 p_{31}| &\leq a_2 \\ |y_1 x_2 + y_2 p_{32}| &\leq a_1 \\ |g_2 x_1 + r_3 p_{31}| &\leq (5/4) a_2 \\ |g_2 x_2 + r_3 p_{32}| &\leq (5/4) a_1 \end{aligned}$$

which, in view of the above estimates, can be seen to hold. All other entries in $P$ are clearly bounded by one of the three numbers. □

This indicates that, by modifying Blankinship's method so that the selected element $a_i$ is used to reduce all other elements, we greatly improve the overall performance of the method, in terms of the size of the final entries. Such an approach is already used successfully in [HHR93]. It seems harder to analyze modifications based on best remainders rather than positive remainders. Another question which needs further study is: in which order should elements of $\mathbf{a}$ be chosen to reduce the other elements? A good approximation scheme that selects $a_i$ in an intelligent manner and guarantees certain performance, in terms of the size of the entries of $P$, could greatly influence the overall performance of Hermite normal form computations.

## 5 Conclusions

If modular methods are not used then Hermite normal form computation for an integer matrix is liable to involve unacceptable intermediate entry explosion. Even polynomial time algorithms, like those of Kannan and Bachem [KB79] or Chou and Colling [CC82], offer relatively poor performance and their implementation requires infinite precision arithmetic for practical examples. On the other hand, as indicated in [HHR93], for many matrices such computations can be done using standard 32 bit arithmetic.

Here we have investigated a possible option for designing an efficient algorithm for Hermite normal form computations. A promising approach is offered by a suitable modification of Blankinship's algorithm [Bla63]. We showed that for three numbers, by using a specific technique, we can achieve close to optimal performance. This type of approach can be extended to an arbitrary number of numbers. A careful study of this technique may lead to efficient integer based methods.

We have developed practical implementations of HNF algorithms consistent with the analyses presented here, which will be described elsewhere. They have very attractive performance, including applications to matrices arising in computational group and number theory. They are based on heuristics which reduce intermediate entry growth in Gaussian elimination over the integers. They outperform previous integer methods on a wide range of examples. We have also tested these ideas in other contexts and they have shown general applicability. Thus, similar methods improve the performance of exact rational Gaussian elimination, and we expect that the principles will extend to other exact matrix computation problems.

## References

[Bla63] W.A. Blankinship. A new version of the Euclidean algorithm. *Amer. Math. Mon.*, 70:742–745, 1963.

[Bod56] E. Bodewig. *Matrix Calculus.* North Holland, Amsterdam, 1956.

[Bra71] G.H. Bradley. Algorithms for Hermite and Smith normal matrices and linear diophantine equations. *Math. Comput.*, 25:897–907, 1971.

[CC82] T-W.J. Chou and G.E. Collins. Algorithms for the solution of systems of linear Diophantine equations. *SIAM J. Comput.*, 11:687–708, 1982.

[Fru76] M.A. Frumkin. An application of modular arithmetic to the construction of algorithms for solving systems of linear equations. *Soviet Math. Dokl.*, 17:1165–1168, 1976.

[Her51] C. Hermite. Sur l'introduction des variables continues dans la théorie des nombres. *J. Reine Angew. Math.*, 41:191–216, 1851.

[HHR93] G. Havas, D.F. Holt, and S. Rees. Recognizing badly presented $Z$-modules. *Linear Algebra and its Applications*, 192:137–163, 1993.

[Hu69] T.C. Hu. *Integer Programming and Network Flows.* Addison-Wesley, Reading, MA, 1969.

[KB79] R. Kannan and A. Bachem. Polynomial algorithms for computing Smith and Hermite normal forms of an integer matrix. *SIAM J. Comput.*, 8:499–507, 1979.

[Lev56] R.J. Levit. A minimum solution to a diophantine equation. *American Math. Mon.*, 63:647–651, 1956.

[MH94] B.S. Majewski and G. Havas. The complexity of greatest common divisor computations. Technical Report TR0296, The University of Queensland, Brisbane, 1994.

[PB74] I.S. Pace and S. Barnett. Efficient algorithms for linear system calculations; part I — Smith form and common divisors of polynomial matrices. *J. of System Science*, 5:403–411, 1974.

[vEB81] P. van Emde Boas. Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Technical Report MI/UVA 81–04, The University of Amsterdam, Amsterdam, 1981.