On the Worst-case Complexity of Integer Gaussian Elimination

Xin Gui Fang George Havas*

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

Abstract

Gaussian elimination is the basis for classical algorithms for computing canonical forms of integer matrices. Experimental results have shown that integer Gaussian elimination may lead to rapid growth of intermediate entries. On the other hand various polynomial time algorithms do exist for such computations, but these algorithms are relatively complicated to describe and understand. Gaussian elimination provides the simplest descriptions of algorithms for this purpose. These algorithms have a nice polynomial number of steps, but the steps deal with long operands. Here we show that there is an exponential length lower bound on the operands for a well-defined variant of Gaussian elimination when applied to Smith and Hermite normal form calculation. We present explicit matrices for which this variant produces exponential length entries. Thus, Gaussian elimination has worst-case exponential space and time complexity for such applications. The analysis provides guidance as to how integer matrix algorithms based on Gaussian elimination may be further developed for better performance, which is important since many practical algorithms for computing canonical forms are so based.

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.

We use the following notation. For a $m \times n$ integer matrix B, say, we denote the entry in the ith row and jth column by $b_{i,j}$. We denote its ith row by $\mathbf{b}_{i*}$ and its jth column by $\mathbf{b}_{*j}$. When we wish to make the dimensions of the matrix clear we denote it by $B_{m \times n}$. The absolute value of x is denoted by $|x|$. We denote by $||B||$ the maximum absolute value of any entry in B. Matrix B may be alternatively written as

$$B = (\mathbf{b}_{*1}, \ldots, \mathbf{b}_{*n}) = \begin{pmatrix} \mathbf{b}_{1*} \\ \vdots \\ \mathbf{b}_{m*} \end{pmatrix}.$$

It follows from a result of Hermite [12] that for any integer matrix B there exists a unique upper triangular matrix H which is row equivalent to B and which satisfies the following conditions.

1. Let r be the rank of B. Then the first r rows of H are nonzero.
2. For $1 \leq i \leq r$ let h_{i,j_i} be the first nonzero entry in row i. Then $j_1 < j_2 < \ldots < j_r$.
3. $h_{i,j_i} > 0$, for $1 \leq i \leq r$.
4. 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 are many algorithms based on Gaussian elimination for computing the HNF. Unfortunately such algorithms suffer from serious practical difficulties. 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 [6], which provides background material also relevant to HNF calculation.

Over the years different strategies have been proposed, primarily trying to avoid the major obstacle that occurs in such computations — explosive growth in size of intermediate entries. A comprehensive bibliography and a number of earlier methods are examined in [6]. More recent methods are described in [7, 5, 19, 18, 10]. In [7, 10] the focus is on finding well-performing algorithms and heuristics for Gaussian elimination methods. Here we look at the other side of the issue: worst case performance for Gaussian elimination. Understanding worst case behaviour is an important step in developing good heuristics to avoid poor performance.

Frumkin [3, 4] has made claims about bounds on the size of intermediate entries which may arise during such computations. However the papers include errors (some of which are typographical) and there are no proofs given.

*partially supported by the Australian Research Council

2 A specific algorithm

Pseudocode for Hermite normal calculation due to Sims [17, p. 323] forms the basis of our specific variant. Figure 1 gives that algorithm rewritten using our notation, with line numbers for easy reference.

```
1  procedure ROWHNF ($B$, $A$);
2  integer $B_{m\times n}$ { input } ; $A_{m\times n}$ { output } ;
3  begin
4  $A := B$; $i := 1$; $j := 1$;
5  while $i \le m$ AND $j \le n$ do
6     { check if rest of column $j$ is zero }
7     if $a_{k,j} = 0$ for $i \le k \le m$ then $j := j + 1$
8     else do
9        while $\exists\ i \le k \ne l \le m$ such that $0 < |a_{k,j}| \le |a_{l,j}|$ do
10          $q := a_{l,j}$ div $a_{k,j}$;
11          $\mathbf{a}_{l\bullet} := \mathbf{a}_{l\bullet} - q \times \mathbf{a}_{k\bullet}$
12       endwhile
13       { $\exists!\ k, i \le k \le m$ such that $a_{k,j} \ne 0$ }
14       interchange $\mathbf{a}_{i\bullet}$ and $\mathbf{a}_{k\bullet}$
15       if $a_{i,j} < 0$ then $\mathbf{a}_{i\bullet} := -\mathbf{a}_{i\bullet}$
16       for $l := 1$ to $i - 1$ do
17          $q := a_{l,j}$ div $a_{i,j}$;
18          $\mathbf{a}_{l\bullet} := \mathbf{a}_{l\bullet} - q \times \mathbf{a}_{i\bullet}$
19       endfor
20    endif
21    $i := i + 1$; $j := j + 1$;
22 endwhile
23 end
```

Figure 1: *Pseudocode for Hermite normal calculation*

During execution of the algorithm there is **enormous** choice in selecting k and l in the while loop on lines 9 to 12. There is an exponential number of different execution sequences possible for a given input matrix depending on the choices made. Furthermore, the size of the intermediate entries depends critically on the choices. Finding optimum choices is NP-hard in a well-defined sense, as reported in [7].

We define one step of the algorithm to comprise the work done in putting one column into final form. (This entails one execution of the code from lines 6 to 21.) Thus the first step creates an equivalent matrix to the input, with first column in HNF form.

To complete the specification of our variant of the algorithm it suffices to indicate how k and l in the key loop, lines 9 to 12, are to be chosen. We do this in the next section, which includes our examples. (The loop in lines 16 to 19 does not affect the analysis.)

3 Analysis

Theorem 1 *Given integers $n > 0$ and $x > 1$ there exists a $2n \times (n+1)$ integer matrix A, with $||A|| = x$ such that the maximal magnitude intermediate entry in the working matrix is greater than x^{2^n} during the nth step of our algorithm.*

For a given integer $x > 1$ we first give the following two constructions, denoted by $A(n, x)$ and $A'(n, x)$, respectively. Matrix A handles the case of even x, while matrix A' handles odd x. We then show that such matrices satisfy all conditions of the theorem.

3.1 Construction A

Let $x > 1$ be even. Set $A(1, x) = \begin{pmatrix} 1 & x \\ x & -1 \end{pmatrix}$. Then inductively define $A(i+1, x)$ by adding a 2-row, 1-column border (which substantially duplicates the previous last row and column but with four new entries):

$$A(i+1,x) = \left(\begin{array}{ccc} & & a_{1,i+1} \\ & A(i,x) & \vdots \\ & & a_{2i,i+1} \\ a_{2i,1},\dots,a_{2i,i}, & (-1)^i & (-1)^{i-1} \\ a_{2i,1},\dots,a_{2i,i}, & 0 & (-1)^i \end{array}\right). \tag{1}$$

Thus

$$A(n,x) = \left(\begin{array}{ccccccc} 1 & x & x & x & x & x & x & \cdots \\ x & -1 & -1 & -1 & -1 & -1 & -1 & \\ x & 1 & -1 & -1 & -1 & -1 & -1 & \\ x & 0 & 1 & 1 & 1 & 1 & 1 & \\ x & 0 & -1 & 1 & 1 & 1 & 1 & \\ x & 0 & 0 & -1 & -1 & -1 & -1 & \\ x & 0 & 0 & 1 & -1 & -1 & -1 & \\ x & 0 & 0 & 0 & 1 & 1 & 1 & \\ x & 0 & 0 & 0 & -1 & 1 & 1 & \\ x & 0 & 0 & 0 & 0 & -1 & -1 & \\ x & 0 & 0 & 0 & 0 & 1 & -1 & \\ x & 0 & 0 & 0 & 0 & 0 & 1 & \\ \vdots & & & & & & & \end{array}\right) \tag{2}$$

3.2 Construction A'

Let $x > 1$ be odd. Set $A'(1, x) = \begin{pmatrix} 1 & x \\ x & -2 \end{pmatrix}$. Then define $A'(i+1, x)$ (again bordered, this time with five new entries) by

$$A'(i+1,x) = \left(\begin{array}{ccc} & & a_{1,i+1} \\ & A'(i,x) & \vdots \\ & & a_{2i-1,i+1} \\ & & (-1)^{i-1} \\ a_{2i,1},\dots,a_{2i,i}, & 0 & (-1)^{i-1} \\ a_{2i,1},\dots,a_{2i,i}, & (-1)^{i-1} & (-1)^i 2 \end{array}\right). \tag{3}$$

Thus

$$A'(n,x) = \left(\begin{array}{cccccc} 1 & x & x & x & x & x & \cdots \\ x & -2 & -1 & -1 & -1 & -1 & \\ x & 0 & -1 & -1 & -1 & -1 & \\ x & -1 & 2 & 1 & 1 & 1 & \\ x & -1 & 0 & 1 & 1 & 1 & \\ x & -1 & 1 & -2 & -1 & -1 & \\ x & -1 & 1 & 0 & -1 & -1 & \\ x & -1 & 1 & -1 & 2 & 1 & \\ x & -1 & 1 & -1 & 0 & 1 & \\ x & -1 & 1 & -1 & 1 & -2 & \\ \vdots & & & & & & \end{array}\right) \tag{4}$$

We now prove the theorem via the following two lemmas.

Lemma 2 *Let $x > 1$ be an even integer. Then Theorem 1 holds for the matrices $A(n, x)$ given as in Equation 1.*

Proof. The proof is by induction. Assume that after the nth step of the algorithm the working matrix has the following form

$$\begin{pmatrix} I_n & * \\ 0 & A^{(n)} \end{pmatrix}.$$

We make this true for $n = 1$ by defining part of our selection method for k and l in the key loop. Thus, if there exists j such that $|a_{n,j}| = 1$ choose $k = j$. (In our matrices there will be at most one such j at any time.) Choose l in an arbitrary fashion till the loop is completed. Thus, for the first column, the $a_{1,1}$ entry is used to set all other entries in that column to zero.

Let m_n be the maximal magnitude entry of $A^{(n)}$. It can be shown by induction on n that $m_n = a^{(n)}_{2n,n+1}$ is odd and is also the unique maximal magnitude entry of $\mathbf{a}^{(n)}_{*(n+1)}$ (part of the $(n+1)$st column of the working matrix). In fact $A^{(n)}$ has the form

$$\begin{pmatrix} \multicolumn{2}{c}{B^{(n)}_{(n-1)\times 2}} & \cdots \\ \pm m_n & \pm m_n & \cdots \\ \pm(m_n-2) & \pm m_n & \cdots \\ \pm(m_n-1) & \pm(m_n-2) & \cdots \\ \vdots & \vdots & \end{pmatrix},$$

where all entries of $B^{(n)}_{(n-1)\times 2}$ are even.

Now we can specify k and l for the first time in the key loop of the $(n+1)$st step of our algorithm. Choose $k = 2n+1$ (corresponding to the entry $\pm(m_n - 2)$) and $l = 2n$ (corresponding to the entry $\pm m_n$). This gives us a row, $\mathbf{a}_{(2n)*}$, with leading entry 2. Now choose $k = 2n$ (corresponding to the entry 2) and $l = 2n+1$ (corresponding to the entry $\pm(m_n - 2)$). This gives us a row, $\mathbf{a}_{(2n+1)*}$, with leading entry 1. Now choose $k = 2n + 1$ (corresponding to the entry 1, as in the first step) for the remainder of this loop to eliminate (in any order) all remaining nonzero entries in this column. Note, in particular, what happens when $l = 2n+2$: the entry in the next column, $a^{(n+1)}_{2n+2,n+2}$, attains the maximal magnitude, $m_{n+1} = \pm(m_n(m_n-1)-(m_n-2))$. Thus

$$m_{n+1} > (m_n - 1)^2.$$

It follows that

$$m_{n+1} > (m_n - 1)^2 > (m_{n-1} - 1)^{2^2} > \ldots > (m_1 - 1)^{2^n}.$$

Since $m_1 = x^2+1$, $m_{n+1} > (x^2)^{2^n} = x^{2^{n+1}}$. So the theorem is true for $A(n, x)$.

Lemma 3 *Let $x > 1$ be an odd integer. Then Theorem 1 holds for the matrices $A'(n,x)$ given as in Equation 3.*

Proof. A similar argument to the proof for Lemma 2 applies. Again we obtain three entries m_n, $m_n - 2$ and $m_n - 1$ in analogous positions in the corresponding matrix and the same kind of explosion occurs.

An explanation of the bad performance we see here comes from consideration of the extended gcd computation which comprises the first part of the calculation for the key loop. Extended gcd calculation is studied in detail in [14, 11, 15, 8, 9, 2, 16]. Here we are computing the gcd of three numbers: $2r + 1$, $2r - 1$ and $2r$. We implicitly construct a solution vector plus a basis for the null-space. It is easy to see that our variant of Gaussian elimination gives as a solution to this problem

$$\begin{pmatrix} -(r-1) & r & 0 \\ 2r-1 & -(2r+1) & 0 \\ 2r(r-1) & -2r^2 & 1 \end{pmatrix} \begin{pmatrix} 2r+1 \\ 2r-1 \\ 2r \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

while an optimal solution is provided by

$$\begin{pmatrix} 1 & 0 & -1 \\ 1 & 1 & -2 \\ -r & r & 1 \end{pmatrix} \begin{pmatrix} 2r+1 \\ 2r-1 \\ 2r \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

The quadratic entries in the last row of the poor solution are propagated through the working matrix.

Corollary 4 *Gaussian elimination has worst-case exponential space and time complexity for Hermite normal form calculation.*

Proof. The size of $A(n,x)$ is $\Omega(n^2 + n \log x)$. Gaussian elimination as described will generate entries in the working matrix with magnitude x^{2^n} during the nth step of the algorithm. These require exponential space to store and exponential time to compute, in terms of the size of the input.

4 Concluding Remarks

We have shown that the worst case behaviour of Gaussian elimination for computing the Hermite normal form of an integer matrix has exponential space and time complexity. This result also applies to general row echelon form computation since we ignored the steps of our algorithm (lines 16 to 19) which normalize the above-diagonal entries of the matrix. Likewise it applies to Smith normal form calculation. (In fact for $n > 1$ the Hermite and Smith normal forms of $A(n,x)$ and $A'(n,x)$ are the same, an $(n+1) \times (n+1)$ identity matrix above $(n-1)$ rows of zeros.)

The immediate cause of the entry explosion comes from an inefficient solution to the extended gcd problem being constructed for triples $2r + 1$, $2r - 1$ and $2r$. It is worth noting that the polynomial-time algorithms of Kannan and Bachem [13] and of Chou and Collins [1], when applied to a column vector with those numbers in that order, also produce the same bad transforming matrix, as does the natural recursive extended gcd algorithm. (The algorithms of Kannan and Bachem and of Chou and Collins then remedy this by reducing off-diagonal entries to attain their polynomial complexity bounds.) However, the improved sorting gcd algorithm [11, 8] produces the optimal transforming matrix. This goes some way to explaining the better performance of integer matrix algorithms based on Gaussian elimination which uses sorting gcd principles. It also provides guidance as to how such algorithms may be further developed for better performance. This is important since many practical algorithms for computing canonical forms are so based, see [6, 7, 10].

References

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

[2] D. Ford and G. Havas. A new algorithm and refined bounds for extended gcd computation. *Algorithmic Number Theory*, Lecture Notes Comput. Sci. **1122** (1996) 145–150.

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

[4] M.A. Frumkin. Complexity questions in number theory. (Russian) *Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov.* **118** (1982) 188–210, 216 (English translation *J. Soviet Math.* **29** (1985) 1502–1517).

[5] M. Giesbrecht. Fast computation of the Smith normal form of an integer matrix. *ISSAC'95* (Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation), ACM Press (1995) 110–118.

[6] G. Havas, D.F. Holt and S. Rees. Recognizing badly presented Z-modules. *Linear Algebra and its Applications* **192** (1993) 137-163.

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

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

[9] G. Havas and B.S. Majewski. A hard problem that is almost always easy. *Algorithms and Computation*, Lecture Notes Comput. Sci. **1004** (1995) 216–223.

[10] G. Havas and B.S. Majewski. Integer matrix diagonalization. *J. Symbolic Computation* (to appear).

[11] G. Havas, B.S. Majewski and K.R. Matthews. Extended gcd algorithms. Technical Report **302**, Department of Computer Science, The University of Queensland (1994).

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

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

[14] B.S. Majewski and G. Havas. The complexity of greatest common divisor computations. *Algorithmic Number Theory*, Lecture Notes Comput. Sci. **877** (1994) 184–193.

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

[16] C. Rössner and J.-P. Seifert. The complexity of approximate optima for greatest common divisor computations. *Algorithmic Number Theory*, Lecture Notes Comput. Sci. **1122** (1996) 307–322.

[17] C.C. Sims. *Computation with finitely presented groups.* Cambridge University Press (1994).

[18] A. Storjohann. Near optimal algorithms for computing Smith normal forms of integer matrices. *ISSAC'96* (Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation), ACM Press (1996) 267–274.

[19] A. Storjohann and G. Labahn. Asymptotically fast computation of Hermite normal forms of integer matrices. *ISSAC'96* (Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation), ACM Press (1996) 259–266.