J. Symbolic Computation (1997) **24**, 399–408

Integer Matrix Diagonalization

GEORGE HAVAS† AND BOHDAN S. MAJEWSKI

Department of Computer Science, The University of Queensland, Queensland 4072, Australia

We consider algorithms for computing the Smith normal form of integer matrices. A variety of different strategies have been proposed, primarily aimed at avoiding the major obstacle that occurs in such computations—explosive growth in size of intermediate entries. We present a new algorithm with excellent performance.

We investigate the complexity of such computations, indicating relationships with NP-complete problems. We also describe new heuristics which perform well in practice. We present experimental evidence which shows our algorithm outperforming previous methods.

1. Introduction

Integer matrices A and B are equivalent if there exist unimodular matrices P and Q such that $PAQ = B$. Matrices P and Q correspond to elementary row and column operations: negating a row (or column); adding an integer multiple of one row (or column) to another; or interchanging two rows (or columns). It follows from a result of Smith (1861) that for any integer matrix there exists a unique equivalent diagonal matrix S, with $s_{ij} = 0$ for $i \neq j$, such that the diagonal entries are nonnegative and s_{i-1i-1} divides s_{ii}. This matrix is called the Smith normal form of the given matrix and has many important applications.

Smith gave a method for computing the Smith normal form, and his approach is used in probabilistic algorithms by Giesbrecht (1995). However it seems unsuitable for deterministic 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 number of methods are examined by Havas *et al.* (1993), who include a comprehensive bibliography. Here we present a new algorithm with significantly better performance.

Modular methods can be used to solve these problems, but the complexity bounds on such computations have relatively high degree when compared to integer based methods. That is why we study integer methods in this paper.

It is often the case that, for a matrix that has quite small entries in both the initial and the Smith normal form, the intermediate entries will be spectacularly large.

† E-mail: havas@cs.uq.edu.au

0747–7171/97/030399 + 10 $18.00/0 sy960141

Frumkin (1976) observed that for simple Gaussian elimination a worst case bound on the length of the entries is exponential. Examples are presented by Havas *et al.* (1993) which show that known polynomially bounded algorithms lead to quite large entries. Consequently, a careless algorithm, while executing a polynomial number of operations, may have exponential complexity simply because of the size of the numbers on which it operates. In this paper we investigate the complexity of such computations, indicating some relationships with NP-complete problems. As a consequence we suggest and justify a new heuristic strategy, with a polynomially bounded number of operations, which performs well in practice.

We use the following notation. For a $m \times n$ integer matrix A we denote its ith row by $\boldsymbol{a}_{i*}$ and its jth column by $\boldsymbol{a}_{*j}$. The absolute value of x is denoted by $|x|$, while $\det(A)$ stands for the determinant of A. Matrix A may be alternatively written as

$$A = [\boldsymbol{a}_{*1}, \ldots, \boldsymbol{a}_{*n}] = \begin{bmatrix} \boldsymbol{a}_{1*} \\ \vdots \\ \boldsymbol{a}_{m*} \end{bmatrix}.$$

2. Previous Methods

A straightforward algorithm for computing the Smith normal form of an integer matrix is as follows. The first stage of the reduction is to compute an equivalent form

$$\left[\begin{array}{c|ccc} d_1 & 0 & \ldots & 0 \\ \hline 0 & & & \\ \vdots & & A' & \\ 0 & & & \end{array}\right]$$

where d_1 divides every entry in the submatrix A'.

If A is the zero matrix we are finished. If not, choose a nonzero entry (the *pivot*) and move it to a_{11} by suitable row and column interchanges, and make it positive.

While there is an entry a_{1j} in the first row not divisible by a_{11}: compute $a_{1j} = a_{11}q + r$; subtract q times the first column from the jth column; interchange the first and jth columns. Do the same with rows and columns interchanged. This may create new nondivisible entries in the row, so keep interchanging the role of rows and columns till no such nondivisible entries remain.

After this, a_{11} divides every entry in its row and column. Subtract suitable multiples of the first column (row) from the other columns (rows) to replace each entry in the first row (column), except a_{11}, by zero. Then we have the correct shape. If the divisibility condition is satisfied, we have finished. If not, there is an entry a_{ij} such that a_{11} does not divide a_{ij}: then add the ith row to the first row and return to the while statement. (In practice many algorithms simply compute a diagonal form first, sorting out divisibility along the diagonal later. In such algorithms this process is delayed to the final step.) Finally, reduce A' recursively.

The algorithm outlined above does not specify how pivots should be chosen, one feature that is exploited in heuristics presented by Havas *et al.* (1993). The general idea there is to consider various mathematical combinations of row and column metrics associated with potential pivots. Thus, pivots are chosen which minimize a combination of associated row and column metrics. Each combination can be interpreted as an estimate of some

local property of the matrix. (As an implementation note we emphasise that care needs to be taken both in calculating the metrics and in combining them to avoid adversely impacting the algorithm. For example, we use fast approximations of the metrics and we use data structures which allow us to find minimal combinations rapidly. Also, after an initial computation, the metrics merely need updating for each row and column in which elements change.)

In the following sections we investigate a different aspect of the basic algorithm. Namely, we assume that we have selected the pivot, and now we want to force a number of entries of A to 0. As we will see, this operation has certain flexibility, which, when properly exploited, can lead to particularly good behavior of the algorithm.

3. Macro-step Analysis

Our primary goal in Smith normal form computations is to minimize the size of the intermediate entry with maximum magnitude. Here we choose to perform an easier task, where we minimize the size of the intermediate entries during each macro-step. By a *macro-step* we mean all of the operations required to force all entries in a row and column to zero, except for the entry on the main diagonal. We need to execute $\min(m, n)$ macro-steps to compute the Smith normal form of a matrix. Minimizing the size of the entries during each macro-step does not guarantee global optimality, since it is conceivable that the calculation should not proceed macro-step by macro-step. However, we will see that even this simplified problem resists good polynomial time solutions.

While investigating algorithms for computing the Smith normal form we have observed that a method with appalling performance is one that looks only at two vectors at a time. This is a natural consequence of the poor performance of this approach in extended gcd computations, discussed in substantial detail by Havas *et al.* (1994) and Majewski and Havas (1995). Thus, given two columns $\boldsymbol{a}_{*1}$ and $\boldsymbol{a}_{*j}$, the method replaces the first column by $x_1\boldsymbol{a}_{*1} + x_2\boldsymbol{a}_{*j}$ and the second column by $a_{1j}/\gcd(a_{11}, a_{1j})\boldsymbol{a}_{*1} - a_{11}/\gcd(a_{11}, a_{1j})\boldsymbol{a}_{*j}$. The multipliers x_1 and x_2 are obtained from $\gcd(a_{11}, a_{1j}) = x_1a_{11} + x_2a_{1j}$. A quick analysis reveals the reason why the performance of this method tends to be bad. If the first q entries, $a_{11}, \ldots, a_{1q}$ are such that $\gcd(a_{11}, \ldots, a_{1p-1}) > \gcd(a_{11}, \ldots, a_{1p}) \neq a_{1p}$, for all $p \leq q$, then the first column after q steps will be

$$\boldsymbol{a}_{*1} \prod_{k=1}^{q-1} x_{2k-1} + \sum_{p=2}^{q} \left(\boldsymbol{a}_{*p} x_{2(p-1)} \prod_{k=p}^{q-1} x_{2k-1} \right)$$

where the atomic multipliers are computed from the equality: $x_{2k-1}\gcd(a_{11}, \ldots, a_{1p-1}) + x_{2k}a_{1p} = \gcd(a_{11}, \ldots, a_{1p})$. The cumulative multipliers (given by the product of the atomic multipliers), as indicated by Bradley (1970), tend to be rather large. Consequently, the expected size of the entries in the first column after the first q elements are forced to zero is substantial. This naturally has a bad impact on the remaining reduction, leading possibly to an unwanted explosion in the size of the intermediate entries. Observe that if a_{11} and a_{12} are relatively prime, as occurs with probability more than 60% for random integers (Knuth, 1973), the average size of the multipliers is $O(a_{12})$ and $O(a_{11})$. As a consequence some entries in the transformed first column can be as large as $O(a_{11}a_{12})$. Using the transformed column in subsequent operations may cause a more than linear increase in the size of new entries, and consequently start a snowballing effect.

Notice that the above method suffers from the narrowness of its horizon. Obtaining

optimal multipliers, that is multipliers such that $|x_1| \leq |a_{1j}|/2$ and $|x_2| \leq |a_{11}|/2$, is all the algorithm can do to minimize the intermediate entries in matrix A. This does not provide enough freedom and, consequently, the size of the intermediate entries grows very fast. The conclusion we draw from the above is that forcing any entry to 0 should be done in a wider context, possibly taking into account all vectors of A.

In this analysis we use some well-performing heuristics for selecting pivot elements. The result of Rose and Tarjan (1978) shows that finding the best method of pivot selection in Gaussian elimination to minimize fill-in is already an NP-complete problem. Also, computing optimal multipliers in extended gcd computations is NP-complete (Majewski and Havas, 1994). It follows that pivot selection to minimize entry size in Smith normal form computation, which is achieved by a form of Gaussian elimination and incorporates extended gcd computations, will be difficult in general.

In each macro-step, one column and row are cleared; i.e., all entries in the column and row (except for the entry on the main diagonal) are forced to zero. The remaining nonzero entry must, once this step is complete, be the greatest common divisor of all other entries in the row and column at the beginning of the step. Hence we consider the following optimization problem.

We are given an integer matrix

$$B = \begin{bmatrix} b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \dots & b_{mn} \end{bmatrix} = [\boldsymbol{b}_{*1}, \boldsymbol{b}_{*2}, \dots, \boldsymbol{b}_{*n}]$$

where $\boldsymbol{b}_{*i}$ is the ith column of B. Find an integer row vector $\boldsymbol{x} = [x_1, \dots, x_m]$, with $x_1 \neq 0$, such that

$$\boldsymbol{x} \cdot \boldsymbol{b}_{*1} = \gcd(b_{11}, \dots, b_{m1})$$

minimizing

$$\max_{i,j} \left| b_{ij} - \frac{\boldsymbol{x} \cdot \boldsymbol{b}_{*j}}{\boldsymbol{x} \cdot \boldsymbol{b}_{*1}} b_{i1} \right|.$$

(The expression being minimized is the largest entry in the new matrix obtained by computing the gcd of the entries in the ith column and then zeroing out all other entries in that column.)

By casting the $(m-i) \times (n-i)$ submatrix of A into B and solving the above problem we can perform a macro-step in the computation of the Smith normal form of A, aiming to achieve optimal or close to optimal performance in terms of the size of the intermediate entries.

An equivalent form of expression for the new value of b_{ij} is

$$b_{ij} \leftarrow \sum_{k=1}^{m} x_k \frac{\det\left(\begin{bmatrix} b_{k1} & b_{kj} \\ b_{i1} & b_{ij} \end{bmatrix}\right)}{\gcd(b_{11}, \dots, b_{m1})}. \tag{3.1}$$

Consider now a single new row of B. Denote by $\boldsymbol{d}_{ij}^{\mathrm{T}} = [d_{ij1}, \dots, d_{ijm}]$ the vector of coefficients associated with the x_ks in (3.1), $d_{ijk} = (b_{k1}b_{ij} - b_{kj}b_{i1})/\gcd(b_{11}, \dots, b_{m1})$. Each b_{ij} can be expressed now as $\boldsymbol{d}_{ij} \cdot \boldsymbol{x}$. Hence the ith row of B, $\boldsymbol{b}_{i*}$, can be computed

as

$$\boldsymbol{b}_{i*} = T_i \boldsymbol{x} = \sum_{q=1}^{n} \boldsymbol{t}_{*q} x_q \tag{3.2}$$

where $T_i = [[t_{pq}]_{p=1}^{m}]_{q=1}^{n}$, with $t_{pq} = d_{ipq}$. The last equation presents a rather unpleasant surprise. The problem of minimizing the maximum entry in $\boldsymbol{b}_{i*}$ where $\boldsymbol{b}_{i*}$ is expressed as a linear combination of n vectors has been proved to be NP-complete by van Emde Boas (1981, see also Majewski and Havas, 1994). In our case the task is even more complicated as we do not have a total freedom in selecting values for the x_is. It is possible to express each x_i as $\sum_j c_{ij} q_j$, for some constants c_{ij} and integer values q_j (cf. Niven *et al.* 1991, §5.2 or Blankinship, 1963), and therefore we can generate any legal set of multipliers; however we cannot "tune" vector $\boldsymbol{x}$, by adjusting any single coordinate without changing the remaining ones. There is a close relationship between the above optimization problem and the NP-complete CLOSEST VECTOR problem of van Emde Boas (1981).

An alternative way to approach the problem of keeping the intermediate entries small during Smith normal form computation might be to choose only a small fixed number of vectors $k > 2$ and use them to perform a single macro-step. Intuitively we must gain something by looking at $k > 2$ vectors at each step instead of just 2. Unfortunately, we do not know how large k should be. This follows from the fact that the gcd of the leading entries of these k vectors must divide all leading entries of the remaining vectors, which is equivalent to the requirement that the gcd of the leading entries of the selected k vectors must be equal to the gcd of the leading entries of all n vectors. Even choosing the smallest k with this property is not an easy task in general, as is shown by the following result (Majewski and Havas, 1994).

THEOREM 3.1. *Given a set of integers U and a positive constant $K \leq |U|$, the task of finding a subset $R \subseteq U$, such that $|R| \leq K$ and* $\gcd(r \in R) = \gcd(u \in U)$ *is NP-complete.*

However this task is easy on average, see Havas and Majewski (1995), indicating that suitable heuristics will perform well almost always.

4. A Greedy Approach

Since the problem of minimizing the intermediate entries during Smith normal form computations is very likely to be intractable, we redirect our attention to methods that, although they cannot guarantee optimal performance all the time, at least perform reasonably well in practice. As a starting point we select the approach described by Havas *et al.* (1993). We show that a single macro-step can be slightly modified, leading to an improvement in the size of the intermediate entries.

First we note that from a theoretical point of view it is quite convenient to consider the process as consisting of two parts: (i) bringing the first entry in the first row to the gcd of the all nonzero entries in the first column; and then (ii) using the first row to force the leading entries to zero. However, in practice this approach requires a gcd method that computes very small multipliers. Observe that in equation (3.1) the size of a new entry in B depends directly on the size of the multipliers. In a separate paper we prove that minimizing the size of the multipliers is intractable (Majewski and Havas, 1994). Consequently this macro-step structure leads directly to hard problems.

A simple alteration however brings quite substantial benefits. Basically, we do differ-

ent quotient-remainder calculation steps, avoiding direct pairwise gcd calculation steps. Consider the method due to Blankinship (1963). Originally, the purpose of the algorithm was to express the gcd of $n \geq 2$ numbers as their linear combination. However the algorithm actually produces a unimodular matrix P such that, for a vector of n integers $\boldsymbol{a} = [a_1, \ldots, a_n]^{\mathrm{T}}$, we have $\sum_{j=1}^{n} p_{1j}a_j = \gcd(a_1, \ldots, a_n)$ and $\sum_{j=1}^{n} p_{ij}a_j = 0$, for $2 \leq j \leq n$. 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 $\boldsymbol{a}$, say a_i. Select any other nonzero element in $\boldsymbol{a}$, 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 $\boldsymbol{a}$ is left. In accordance with common practice we select the lowest available i and j and use positive remainder in our quotient-remainder computations. This adds up to doing repeated pairwise gcd calculations.

At each step the operator row is used to reduce only one row. If, instead, the appropriate multiples of the operator row are subtracted from all other rows then the resulting matrix has much smaller entries.

Example 4.1. (Havas and Majewski, 1994) *Consider the following vector* $[f_n, f_{n+1}, f_{n+2} - 1]^{\mathrm{T}}$*, where* f_i *is the* i*th Fibonacci number. Blankinship's algorithm for this vector leads to the following equation:*

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

Observe that entries in the third row of the transforming matrix are quite large. Now suppose we perform the same basic sequence of the operations, however we subtract multiples of the operator row from both other rows. In this case it means that during Blankinship's odd numbered steps we subtract the first row from the second **and third**, *and during the (unchanged) even numbered steps we subtract the second row from the first. As a result we obtain:*

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

Notice the big reduction in the size of the entries in the last row. Even better performance is obtained when we use the best remainder strategy of Havas et al. (1993). *Then we obtain:*

$$\begin{bmatrix} 1 & 1 & -1 \\ f_{n-4} - 5 & f_{n-4} + 3 & -f_{n-4} \\ f_{n-2} - 2 & f_{n-2} + 1 & -f_{n-2} \end{bmatrix} \begin{bmatrix} f_n \\ f_{n+1} \\ f_{n+2}-1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

The advantage of using the operator row on all rows instead of just one is quite pronounced. □

It is possible to analyze extended gcd algorithms theoretically. One such algorithm is studied in detail by Majewski and Havas (1995). Here we simply observe that if we apply a Blankinship-type algorithm to the first column of matrix B (i.e., we set $a_i = b_{i1}$) then the transforming matrix P that we obtain is a unimodular matrix P with the property

that PB has the same shape as B after a column macro-step, as defined in Section 3. Then a combination of all the above with the results presented by Havas *et al.* (1993) leads to the following heuristic algorithm.

Input: an $m \times n$ matrix (B).

Output: a matrix equivalent to the input matrix with $b_{11} \neq 0$ and $b_{i1} = b_{1j} = 0$, for $i = 2, \ldots, m$ and $j = 2, \ldots, n$ (or the zero matrix).

Method: If all elements of B are equal to zero, then stop. Otherwise, choose such a nonzero element of the matrix B for which the product of the Euclidean column and row norms is minimal. If there is more than one such element, choose one with minimal absolute value. Move the selected element to position $(1, 1)$ by row and column interchanges. This is a pivot selection method which differs from earlier methods in that the primary emphasis is on the row and column metrics rather than on the pivot size. The aim remains to restrain entry growth, and this emphasis gives better performance. Unit pivots are still implicitly preferred since they contribute less to both the row and column norms than nonunit pivots. Further, this heuristic remains fully functional in the absence of unit pivots.

For each nondiagonal entry in the first column (b_{i1}, $i > 1$) compute a "least-remainder" $r_i = b_{i1} - q_i b_{11}$ of this element divided by b_{11}, with integer quotient q_i and such that $|r_i| \leq b_{11}/2$. Repeat this process with the entries in the first row, computing $r'_j = b_{1j} - q'_j b_{11}$. Notice that here we implement the suggestion in Havas *et al.* (1993) to choose a least-remainder rather than more simply taking an arbitrary remainder in the larger range $[-(b_{11} - 1), b_{11} - 1]$. The beneficial effect of this decision in normal gcd calculations was proved by Kronecker (1901, p. 118) and is quantified by Rosen (1992, p. 67). It extends to our computations in a natural way.

For $i = 2, \ldots, m$ subtract q_i times row one from row i, and likewise subtract q'_j times column one from column j. If all elements in the first row and column are zero, then stop. (Note that unit pivots lead to fast stopping.) Otherwise, search through the first column and row for the best new pivot, in terms of the product of the row and column norms. (The search is restricted to the first row and column for three reasons: it is faster than considering the whole matrix; good pivots can be expected to be found in this row or column because the initial pivot was good; and standard algorithm termination proofs rely on it.) As before, if there is more than one potential pivot with the same norm product, choose one of minimal magnitude. Move the new pivot to b_{11} and keep repeating the process till it stops.

Notice that this algorithm is symmetric in the following sense. Once the new pivot has been selected, both row and column reductions are done. However, the order in which row and column reductions are executed does not matter, as the results are the same. Thus, for any submatrix consisting of four elements

$$\begin{bmatrix} b_{11} & b_{1i} \\ b_{j1} & b_{ji} \end{bmatrix}$$

for $j, i \geq 2$, element b_{ji} is set to $b_{ji} - q_1 b_{1i} - q_2 r_1$, if row reduction precedes column reduction, and is set to $b_{ji} - q_2 b_{j1} - q_1 r_2$ otherwise, where $b_{j1} = q_1 b_{11} + r_1$ and $b_{1i} = q_2 b_{11} + r_2$. This proves that the new value of b_{ji} is the same, regardless of the relative order of the reductions. The correctness of the algorithm follows from the fact that the pivot is selected to be nonzero and the algorithm has to stop after at most $\log b_{11}$

reductions. (The new pivot is always selected from elements that have no more than half the magnitude of the old pivot.)

5. Performance

We have undertaken extensive analyses of the behaviour of the new algorithm. To exemplify this we start by using test data described in detail by Havas *et al.* (1993). This allows us to compare readily the performance of the new heuristics with previous methods.

Our first example is matrix R_1, a 26×27 matrix which arises from a knot group. It is an interesting test case: the initial matrix has rank 25, small entries and moderate density; its Smith normal form has one nontrivial entry, 3. (The matrix comes from abelianizing a presentation for a subgroup of index 13 in a knot group presented on 3 generators and 2 relators. R_1 has 702 entries, 326 of which are zero, 281 of unit magnitude, 91 of magnitude 2, and 4 of magnitude 3.) The full matrix is:

2	0	−1	0	−1	2	1	−1	2	−1	−1	0	−1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	−1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
2	1	−1	0	0	1	0	0	0	0	1	0	1	−1	−1	−1	0	−1	0	1	−1	0	0	0	0	−1
2	0	−1	2	−1	0	−1	1	0	0	0	0	−2	2	0	0	0	0	1	−2	1	1	−1	1	−1	0	0
1	−1	1	0	0	0	−1	0	−1	1	0	0	0	1	1	−1	0	0	0	0	2	−1	0	1	2	0	−1
0	0	0	2	−2	1	1	−2	−1	1	1	0	0	0	−1	0	1	−1	1	0	0	0	2	0	0	−1	0
0	0	0	0	1	0	1	−1	1	−1	−1	−1	−1	−1	0	1	1	0	0	0	0	−1	0	−1	−1	1	0
0	0	0	1	−2	0	2	−2	0	1	2	0	1	0	2	0	−2	1	1	−1	−1	−1	1	1	−1	0	0
2	2	1	−2	1	1	0	0	0	0	1	0	1	−2	0	0	−1	−1	0	1	−2	0	0	−1	0	1	−2
1	1	0	0	−1	2	−1	0	2	−1	−1	−1	−1	−1	1	−1	0	0	1	0	1	0	−1	1	0	0	−1
1	−2	−1	1	1	0	1	−1	0	0	−2	−1	0	1	0	0	1	1	1	−2	1	−1	0	1	0	1	1
1	0	1	1	−1	0	−3	1	0	0	0	0	−2	−1	2	−2	0	0	1	1	2	−1	−1	1	1	0	−1
0	0	0	1	−2	0	−1	1	1	0	1	0	0	0	2	0	−2	1	1	−2	0	1	−1	1	−1	1	0
1	0	0	0	0	0	0	0	−1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	0	0	−2	2	0	0	1	0	1	0	−1	−2	0	0	0	0	1	0	1	0	0	0	0	−1	−2
1	0	−1	0	−1	1	0	1	1	0	0	0	−2	−1	1	0	0	0	1	−1	1	−1	−1	2	−1	0	0
1	1	−1	0	0	1	0	0	0	0	1	0	1	−1	−1	−1	0	0	0	1	−1	0	0	0	0	0	−1
2	−1	0	−1	1	0	0	0	0	−1	−1	0	−1	−1	0	−1	0	−1	0	1	2	−1	−1	0	1	−1	0
1	2	1	−1	0	1	−1	1	1	0	1	0	−2	−1	1	2	0	−1	1	0	2	0	0	−1	1	0	−2
2	1	0	−1	0	1	0	−1	−1	0	0	0	−2	0	−1	−1	1	−1	0	2	2	0	0	−1	1	−2	−1
0	0	0	0	0	0	0	0	1	0	−1	0	0	0	0	−2	0	1	0	−2	−1	0	−1	2	−2	0	1
0	0	−1	0	−1	1	2	−2	0	0	−1	0	−1	−1	2	0	0	2	1	−2	2	−2	0	2	0	0	0
1	1	0	1	−2	2	2	−2	2	−1	0	0	−2	−1	0	1	1	−1	0	0	1	0	2	−1	−1	−1	−1
2	−1	0	0	0	0	0	0	1	−1	1	0	−1	−1	−1	0	0	−3	0	1	1	0	0	−1	0	0	0
1	0	0	0	0	−1	−2	−1	−3	2	1	1	0	2	1	−2	0	0	−1	2	1	0	−1	−2	2	0	−1
2	0	0	1	−2	0	0	−1	−1	0	1	1	−2	2	1	−1	−1	0	−1	1	2	0	0	−1	1	−1	−1

In the studies included by Havas *et al.* (1993) various maximum magnitude entries are reported, depending on strategy used. Thus, after 12 macro-steps using the pivot on maximum strategy, the largest entry already has 1626 decimal digits. The computation completes with a 265-digit maximum entry when the pivot is chosen to be the first nonzero entry. This improves to 110 digits when pivoting on the minimum entry. The polynomial time algorithm due to Kannan and Bachem (1979) produces an 11-digit maximum entry. The metric-based heuristics introduced by Havas *et al.* (1993) achieve maxima between 8501 and 800 722. Our new heuristics outperform the best of these, producing 7269 as maximum.

Our second example is matrix A_1, a 304×153 matrix, sparse and with small entries.

In this case the Kannan–Bachem algorithm produces a 35-digit maximum entry and the metric-based strategies only sometimes succeed in 32-bit precision. The best result without special reduction techniques has 1895 as maximum. Here the new method achieves a maximum of 190. In the past this kind of result has only ever been achieved with very substantial use of lattice basis reduction.

Our third example comes from the index 120 subgroup in the Heineken group. Even with metric-based strategies, previous methods all fail without extra lattice basis reduction steps. The new method succeeds with maximal magnitude entry 10 606.

In all of these cases the new heuristics outperform the best heuristic method presented by Havas *et al.* (1993). While the comparison with the best strategy of Havas *et al.* is not always very impressive, this greedy approach definitely improves on their $\times_2$ method, which uses the same norm to control its decisions. Naturally, as for any heuristic, we can expect that choice of a different metric could lead to an improvement for specific inputs. However, in general, deciding the best metric a priori is difficult. This is the reason we use only one measure, regardless of the input.

In addition, we tested our new method on a number of artificial matrices generated in the following way. First we create a diagonal matrix in its desired Smith normal form. Next we use a scrambling program which, with probability 0.6, negates a randomly selected row and, with probability 0.4, adds to a randomly selected row a constant multiple of another randomly selected row. The constant is randomly selected in the range $[-3, +3]$. The number of times these operations are performed on the input matrix is randomly selected to be between 97 and 154. (The choice of parameters here is dictated by the desire to avoid overflow of 32-bit integers during the scrambling operations.) Matrices created this way were then used as input to the new method. Observe that, as we use only row operations in creating these matrices, it should be (and often is) easier to obtain the Smith normal form by performing row operations before any column operations. However, as it is designed for general case use, the new method does not take advantage of this.

One indicative example is provided by a 10×10 matrix, called A_{21}, whose Smith normal form has the following diagonal entries: 1, 1, 1, 1, 6, 6, 60, 60, 60, 180. Only four of the ten diagonal entries are equal to 1, which means that a Smith normal form computation will run out of unit pivots after the first 4 macro-steps. The scrambled matrix has an initial maximum magnitude entry $-10\,620$. The new method applied to A_{21} gives a maximum magnitude entry of 13 320 after the first macro-step. After the third macro-step the maximum increases to 19 140. The remaining steps do not introduce any larger entries and 19 140 is the largest intermediate entry generated by the algorithm. Havas *et al.* (1993) noted that a potential candidate for proving exponential entry growth is the heuristic that always selects the maximum element as the next pivot. This heuristic, when run on A_{21}, generates a stunning maximum entry that is 11 310 decimal digits long. Pivoting on the first nonzero entry generates a largest entry which is 30 digits long. The Kannan–Bachem algorithm generates a nine decimal digit number. The $\times_2$ strategy of Havas *et al.* (1993) gives a six digit maximal magnitude entry, 101 880.

We also discovered that, for a sample of randomly generated matrices (which frequently have a Smith normal form consisting of $m-1$ units followed by a large number), the new heuristics never lead to an entry larger than the largest number in the Smith normal form. Thus, in many cases we have observed the computation proceeding close to optimally.

6. Concluding Remarks

We have presented a new algorithm for computing the Smith normal form of an integer matrix. The algorithm is based on heuristics which reduce intermediate entry growth in Gaussian elimination over the integers. Practical tests indicate very good performance, outperforming previous integer methods on a wide range of examples.

We have also tested these ideas in other contexts. They have shown general applicability elsewhere. Similar methods reduce the difficulty of Hermite normal form computation for integer matrices (Havas and Majewski, 1994) and exact rational Gaussian elimination. We expect that the principles will extend to many exact matrix computation problems.

Acknowledgement

The authors were supported by the Australian Research Council.

References

Blankinship, W.A. (1963). A new version of the Euclidean algorithm. *Amer. Math. Monthly* **70**, 742–745.
Bradley, G.H. (1970). Algorithm and bound for the greatest common divisor of n integers. *Commun. ACM* **13**, 433–436.
Frumkin, M.A. (1976). An application of modular arithmetic to the construction of algorithms for solving systems of linear equations. *Soviet Math. Dokl.* **17**, 1165–1168.
Giesbrecht, M. (1995). Fast computation of the Smith normal form of an integer matrix. In A.H.M. Levelt, (ed.), *ISSAC'95* (Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation). New York: ACM Press, pp. 110–118.
Havas, G., Holt, D.F., Rees, S. (1993). Recognizing badly presented Z-modules. *Linear Algebra Appl.* **192**, 137–163.
Havas, G., Majewski, B.S. (1994). Hermite normal form computation for integer matrices. *Congressus Numerantium* **105**, 87–96.
Havas, G., Majewski, B.S. (1995). A hard problem that is almost always easy. *Algorithms and Computation*, LNCS **1004**, 216–223.
Havas, G., Majewski, B.S., Matthews, K.R. (1994). Extended gcd algorithms. Technical Report **302**, Department of Computer Science, The University of Queensland.
Kannan, R., Bachem, R. (1979). Polynomial time algorithms for computing Smith and Hermite normal forms of an integer matrix. *SIAM J. Comput.* **8**, 499–507.
Knuth, D.E. (1973). *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Reading: Addison-Wesley, 2nd edition.
Kronecker, L. (1901). *Vorlesung über Zahlentheorie I*. Liepzig: Teubner.
Majewski, B.S., Havas, G. (1994). The complexity of greatest common divisor computations. *Algorithmic Number Theory*, LNCS **877**, 184–193.
Majewski, B.S., Havas, G. (1995). A solution to the extended gcd problem. In A.H.M. Levelt, (ed.), *ISSAC'95* (Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation). New York: ACM Press, pp. 248–253.
Niven, I., Zuckerman, H.S., Montgomery, H.L. (1991). *An Introduction to the Theory of Numbers*. John Wiley & Sons, Inc., 5th edition.
Rosen, K.H. (1992). *Elementary Number Theory and its Applications*. Reading: Addison-Wesley, 3rd edition.
Rose, D.J., Tarjan, R.E. (1978). Algorithmic aspects of vertex elimination on directed graphs. *SIAM J. Appl. Math.* **34**, 176–197.
Smith, H.J.S (1861). On systems of linear indeterminate equations and congruences. *Philos. Trans. Royal Soc. London* **cli**, 293–326. Reprinted in *The Collected Mathematical Papers of Henry John Stephen Smith*, Volume 1. New York: Chelsea (1965).
van Emde Boas, P. (1981). 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.

Originally received 1 January 1995
Accepted 1 January 1995