ELSEVIER Linear Algebra and its Applications 282 (1998) 25–45

LINEAR ALGEBRA AND ITS APPLICATIONS

Computing Hermite and Smith normal forms of triangular integer matrices [1]

Arne Storjohann [2]

Institut für Wissenschaftliches Rechnen, ETH-Zentrum, CH-8092 Zürich, Switzerland

Received 13 October 1997; received in revised form 7 April 1998

Submitted by M. Goldberg

Abstract

This paper considers the problem of transforming a triangular integer input matrix to canonical Hermite and Smith normal form. We provide algorithms and prove deterministic running times for both transformation problems that are optimal in the matrix dimension. The algorithms are easily implemented, assume standard integer arithmetic, and admit excellent performance in practice. The results presented here lead to a faster algorithm for computing the Smith normal form of an arbitrary (i.e. non-triangular) input matrix.

Keywords: Hermite normal form; Smith normal form; Integer matrix

1. Introduction

It follows from Hermite [1] that any $m \times n$ rank n integer matrix A can be transformed by applying a sequence of integer row operations to an upper triangular matrix H that has off-diagonal entries nonnegative and with magnitude smaller than the positive diagonal entry in the same column. The triangularization H – called the Hermite normal form of A – always exists and is unique. In this paper we consider the problem of computing the Hermite normal form H

[1] This work has been supported by grants from the Swiss Federal Office for Education and Science in conjunction with partial support by ESPRIT LTR Project no. 20244 – ALCOM-IT.

[2] E-mail: storjoha@inf.ethz.ch.

0024-3795/98/$19.00 © 1998 Elsevier Science Inc. All rights reserved.
PII: S0024-3795(98)10012-5

of an $n \times n$ non-singular matrix T that is already upper triangular and has off-diagonal entries bounded in magnitude by the product D of the diagonal entries. We get the following transformation diagram

$$\overset{T}{\begin{bmatrix} h_1 & t_{12} & t_{13} & & t_{1n} \\ & h_2 & t_{23} & \cdots & t_{2n} \\ & & h_3 & & t_{3n} \\ & & & \ddots & \vdots \\ & & & & h_n \end{bmatrix}} \rightsquigarrow \overset{H}{\begin{bmatrix} h_1 & \bar{h}_{12} & \bar{h}_{13} & & \bar{h}_{1n} \\ & h_2 & \bar{h}_{23} & \cdots & \bar{h}_{2n} \\ & & h_3 & & \bar{h}_{3n} \\ & & & \ddots & \vdots \\ & & & & h_n \end{bmatrix}}.$$

The diagonal entries h_j are positive, off-diagonal entries t_{ij} of T are bounded in magnitude by $D = h_1 h_2 \cdots h_n$ and off-diagonal entries $\bar{h}_{ij}$ of H satisfy $0 \leqslant \bar{h}_{ij} < h_j$ for $1 \leqslant i < j \leqslant n$. Recall that an integer row operation is one of: (r1) adding an integer multiple of one row to a different row; (r2) switching two rows; (r3) negating a row. For the problem of transforming T to H it will be sufficient to use (r1). The following example shows the transformation of a 3×3 matrix T to Hermite form H.

$$T = \begin{bmatrix} 5 & 2342 & 1843 \\ & 78 & 8074 \\ & & 32 \end{bmatrix}$$

$$\downarrow$$

Subtract 30 times row two from row one.

$$\downarrow$$

$$\begin{bmatrix} 5 & 2 & -240377 \\ & 78 & 8074 \\ & & 32 \end{bmatrix}$$

$$\downarrow$$

Subtract 252 times row three from row two.

$$\downarrow$$

$$\begin{bmatrix} 5 & 2 & -240377 \\ & 78 & 10 \\ & & 32 \end{bmatrix}$$

$$\downarrow$$

Add 7511 times row three to row one.

$$\downarrow$$

$$H = \begin{bmatrix} 5 & 2 & 7 \\ & 78 & 10 \\ & & 32 \end{bmatrix}$$

Since the Hermite form is a row equivalent form, algorithms that solve problems concerned with the *lattice* of T – the set of all integer linear combinations of rows of T – can work with the Hermite form H instead of T itself. The Hermite form has some important advantages. First, T requires $O(n^2 \log D)$ bits to write down but the total size of H will bounded by only $O(n \log D)$ bits; this smaller size translates into improved running times for algorithms that use H instead of T. An example of such a problem is to determine whether or not an n-dimensional integer row vector is contained in the lattice of T. Second, to determine if two integer matrices generate the same lattice, it is sufficient to compare their canonical Hermite forms. This check for row equivalence is not possible with a general (non-unique) triangularization.

To give complexity results we will use the parameters ϵ and θ. A single arithmetic operation with integers bounded in length by t bits costs $O(t^{1+\epsilon})$ bit operations and two $n \times n$ matrices over a ring can be multiplied in $O(n^\theta)$ ring operations. The pseudo-linear algorithm of Schönhage and Strassen [2] allows any fixed ϵ with $\epsilon > 0$ and the current record for θ is $\theta < 2.376$ due to Coppersmith and Winograd [3]. All algorithms presented in this paper assume the standard, eminently practical algorithms for integer and matrix multiplication which have $\epsilon = 1$ and $\theta = 3$.

The Hermite form algorithms in [4–6] all require $O(n^3)$ arithmetic operations with integers bounded in length by $O(\log D)$ bits to reduce T to H. One approach for improving this complexity is to incorporate fast matrix multiplication techniques. This is done in [7] to get a reduced complexity of $O(n^\theta)$ arithmetic operations with integers of the same length. Thus, the previously fastest algorithm to compute H from T requires $O(n^{2.376}(\log D)^{1+\epsilon})$ bit operations assuming asymptotically fast (but currently impractical) matrix multiplication. The algorithm we give here computes H from T in only $O(n^2 \log^2 D)$ bit operations – this is now optimal in n being a factor of only $O(\log D)$ more than the number of bits required to write down the input matrix. Moreover, the algorithm assumes the standard, practical algorithms for integer and matrix multiplication and admits exceilent performance in practice.

The second problem we consider, and the main topic of this paper, is integer matrix diagonalization. It follows from Smith [8] that any $n \times m$ integer matrix A can be transformed by applying a sequence of integer row *and* column operations to a diagonal matrix $S = \mathrm{diag}(s_1, \ldots, s_r, 0, \ldots, 0)$, where r is the rank of A, each s_i is positive, and s_i divides s_{i+1} for $1 \leqslant i \leqslant r-1$. The matrix S – called the Smith form of A – always exists and is unique. For some historical remarks and applications of the Smith form we refer to Newman [9]. In this paper we consider the problem of transforming to Smith form an $n \times m$ matrix T which has principal $n \times n$ submatrix non-singular, upper triangular and all off-diagonal entries bounded in magnitude by the product D of the diagonal entries. We get the following transformation diagram:

$$T \qquad\qquad S$$

$$\begin{bmatrix} t_1 & t_{12} & t_{13} & & t_{1n} & t_{1*} & & t_{1m} \\ & t_2 & t_{23} & \cdots & t_{2n} & t_{2*} & \cdots & t_{2m} \\ & & t_3 & & t_{3n} & t_{3*} & & t_{3m} \\ & & & \ddots & \vdots & \vdots & & \vdots \\ & & & & t_n & t_{n*} & & t_{nm} \end{bmatrix} \rightsquigarrow \begin{bmatrix} s_1 & & & & \\ & s_2 & & & \\ & & s_3 & & \\ & & & \ddots & \\ & & & & s_n \end{bmatrix}.$$

The problem is to transform T to Smith form S using a sequence of integer row and column operations. Note that the total size of T is $O(nm \log D)$ bits. In this paper we prove the surprising result that the Smith form S of T can be computed in only $O(nm \log^2 D)$ bit operations using standard integer arithmetic.

Most previous normal form algorithms have been presented for the case of a non-triangular input matrix A. For brevity let us consider here the case of an input matrix A that is square non-singular, that is, with $n = m$. Let us also assume that we have the quantity D (which will be $|\det A|$) and that entries in A are bounded in magnitude by D. Hafner and McCurley's [5] asymptotically fast triangularization algorithm requires $O(n^{\theta}(\log D)^{1+\epsilon})$ bit operations to produce a triangularization T of A but the algorithm they propose for computing S from T (or directly from A) requires $O(n^3(\log D)(\log D)^{1+\epsilon})$ bit operations – note the extra $O(\log D)$ factor in this complexity result. Giesbrecht [10] has given a Las Vegas probabilistic algorithm that requires about $O(n^3(\log D)^{1+\epsilon})$ bit operations – essentially using randomization to remove this offending $O(\log D)$ factor. Most recently, we have discovered a deterministic algorithm that requires $O(n^{\theta}(\log D)^{1+\epsilon})$ bit operations [11]. Thus, the previously fastest algorithm for computing S from T requires $O(n^{2.376}(\log D)^{1+\epsilon})$ bit operations assuming asymptotically fast matrix multiplication. The algorithm we give here requires only $O(n^2 \log^2 D)$ bit operations using standard arithmetic to compute S from T – this is now optimal in the matrix dimension, being a factor of only $O(\log D)$ more than the size of the input matrix. In the case where $m > n$ we prove a complexity of $O(nm \log^2 D)$ bit operations – also optimal in the matrix dimension. This algorithm is easy to implement in a Computer Algebra system and runs extremely fast in practice.

We apply our triangular Smith normal form algorithm to get a fast, practical and deterministic algorithm for computing the Smith normal form of an arbitrary (i.e. non-triangular) $n \times m$ input matrix A. The cost of the algorithm is $O(nm^3 \log^2 m\|A\| + m^4 \log^3 m\|A\|)$ bit operations where $\|A\| = \max |A_{ij}|$. A faster randomized algorithm is given in [10] that requires about $O(nm^{\theta} \log m\|A\| + m^3 \log^2 m\|A\|)$ bit operations, also assuming standard integer arithmetic, to return a Smith form which is correct with controllable, exponentially small probability of error. The fastest known algorithm which guarantees correctness of the output is the deterministic algorithm in [11].

Assuming asymptotically fast integer and matrix multiplication the algorithm in [11] computes the Smith form S of A within a poly-logarithmic factor of the time required by the currently fastest known algorithm to compute only the determinant of A. Under the assumption of standard integer and matrix multiplication, though, a direct application of the algorithm in [11] requires $O(nm^4 \log^2 m\|A\|)$ bit operations.

The rest of this paper is organized as follows. In Sections 2 and 3 we present our algorithm for transforming an upper triangular input matrix to Hermite and Smith form respectively. In Section 4 we give an algorithm for computing the Smith form of an arbitrary integer input matrix. In Section 5 we conclude with some additional comments about the problem of computing Smith forms of integer matrices. Before continuing we define the complexity model used for the analysis of algorithms in this paper.

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

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

Using standard arithmetic, a and b can be multiplied in $O((\lg a)(\lg b))$ bit operations, and we can express $a = qb + r$, with $0 \leqslant |r| < |b|$, in $O((\lg a/b)(\lg b))$ bit operations. The greatest common divisor $g = \gcd(a, b)$ of a and b can be found in $O((\lg a)(\lg b))$ bit operations; in the same running time we can recover s and t satisfying $sa + tb = g$ with $s \leqslant |b/g|$ and $|t| \leqslant |a/b|$. Finally, each direction of the isomorphism implied by the Chinese remainder algorithm can be computed in $O((\lg N)^2)$ bit operations if N is the product of all the moduli. This complexity model was popularized by Collins [12] and is sometimes called "naive bit complexity" (see, for example, [13]).

2. Hermite normal forms of triangular matrices

Work on this problem was motivated by an asymptotically fast triangularization algorithm of Hafner and McCurley [5] that takes as input an $m \times n$ rank n input matrix and returns as output a row equivalent upper triangular matrix with all entries non-negative and with off-diagonal entries bounded in magnitude by the product D of the diagonal entries – the matrix T of (1) is an example of the principal $n \times n$ submatrix of such a triangularization.

$$\overset{T}{\begin{bmatrix} 8 & 11286 & 4555 & 46515 \\ & 1 & 66359 & 153094 \\ & & 9 & 43651 \\ & & & 77 \end{bmatrix}} \rightsquigarrow \overset{H}{\begin{bmatrix} 8 & 0 & 1 & 51 \\ & 1 & 2 & 20 \\ & & 9 & 69 \\ & & & 77 \end{bmatrix}}. \tag{1}$$

The classical algorithm for transforming T to Hermite form works by placing, for $r = 1, 2, \ldots, n$, the principal rth submatrix of T into correct form. At stage r appropriate multiples of the rth row of T are added to rows $1, 2, \ldots, r-1$ to reduce modulo the diagonal entry the entries in column r.

Classical Triangular Reduction

for $r = 1$ to n do
  for $i = 1$ to $r - 1$ do
    $\text{row}(T, i) = \text{row}(T, i) - \lfloor T_{i,r}/T_{r,r} \rfloor \text{row}(T, r)$;

The classical approach requires $O(n^3)$ arithmetic operations but does not properly bound the magnitudes of intermediate integer entries. The Hermite form algorithms given in [4–6] essentially follow the classical approach but perform all integer operations modulo D; this leads directly to a complexity of $O(n^3)$ arithmetic operations with integers bounded in length by $O(\log D)$ bits. Fast matrix multiplication techniques can be used to reduce the complexity to $O(n^\theta)$ arithmetic operations with integers of the same length [7]. At first sight this appears to be near-optimal in the parameter n since the problem seems tantamount to performing a "matrix operation" [3] on an $n \times n$ matrix. This assessment is false. The algorithm we give here computes H in only $O(n^2 \log^2 D)$ bit operations using standard arithmetic.

Our algorithm is based on an idea due to Chou and Collins [14]; their approach for reducing the off-diagonal entries differs from the classical approach primarily with respect to the order of operations. The algorithm works by transforming, for $r = n, n-1, \ldots, 1$, the *trailing* $(n-r) \times (n-r)$ submatrix of T into correct form. At stage r, appropriate multiples of rows $r+1, r+2, \ldots, n$ of T are added to row r to reduce modulo the diagonal entry in the same column the off-diagonal entries in row r.

Chou and Collins Triangular Reduction

for $r = n$ by -1 to 1 do
  for $j = r + 1$ to n do
    $\text{row}(T, r) = \text{row}(T, r) - \lfloor T_{r,j}/T_{j,j} \rfloor \text{row}(T, j)$;

In [14] this special reduction order was used *during* the transformation of a non-triangular input matrix to Hermite normal form and allowed the authors to obtain a better bound on the size of intermediate integer entries. Here, we combine the reduction order due to Chou and Collins with the modular deter-

[3] Note that the classical triangular reduction algorithm performs $O(n^2)$ row operations on an $n \times n$ matrix – the same as required by the classical algorithms for Gaussian elimination, determinant computation, inversion, etc.

minant approach (cf. [4–6]). A careful analysis of the ensuing algorithm enables us to prove a complexity of only $O(n^2 \log^2 D)$ bit operations. To get this complexity result, we need two refinements to the algorithm. First, all integer operations must be performed modulo d for some integer d bounded by D. The following lemma shows that reducing modulo $d = |t_2 t_3 \cdots t_k|$ any entry in the first row of the following upper triangular, non-singular matrix

$$T = \left[\begin{array}{c|cccc} t_1 & t_{12} & t_{13} & \cdots & t_{1k} \\ \hline & t_2 & t_{23} & \cdots & t_{2k} \\ & & t_3 & & t_{3k} \\ & & & \ddots & \vdots \\ & & & & t_k \end{array}\right]$$

is equivalent to performing a sequence of integer row operations on T.

Lemma 1. *Let T be a $k \times k$ non-singular upper triangular integer matrix and let $d = \prod_{2 \leqslant i \leqslant k} |T_{i,i}|$. For j with $2 \leqslant j \leqslant k$, the matrix T_d obtained from T by reducing modulo d the entry in row 1 column j is row equivalent to T.*

Proof. Let T_1 be the trailing $(k-1) \times (k-1)$ submatrix of T. We claim that the k-dimensional row vector with all entries zero except with jth entry equal to $\det T_1$ can be expressed as an integer linear combination of the last $k-1$ rows of T. To see this, note that $T_1^{\mathrm{adj}} T_1 = \det(T_1) I_{k-1}$ where T_1^{adj}, the adjoint of T_1, is an integer matrix. Since $d = \pm \det T_1$, there exists some integer linear combination of the last $k-1$ rows of T which, when added to row one of T, has precisely the effect of reducing modulo d the entry in column j. □

For the second refinement, we note that some row operations may be sparse. Consider the following $k \times k$ input matrix

$$T = \left[\begin{array}{c|ccccccccc} h_1 & t_{12} & t_{13} & t_{14} & t_{15} & t_{16} & t_{17} & \cdots & t_{1k} \\ \hline & h_2 & & & \bar{h}_{25} & & \bar{h}_{27} & & \bar{h}_{2k} \\ & & 1 & & \bar{h}_{35} & & \bar{h}_{37} & & \bar{h}_{3k} \\ & & & 1 & \bar{h}_{45} & & \bar{h}_{47} & \cdots & \bar{h}_{4k} \\ & & & & h_5 & & \bar{h}_{57} & & \bar{h}_{5k} \\ & & & & & 1 & \bar{h}_{67} & & \bar{h}_{6k} \\ & & & & & & h_7 & & \bar{h}_{7k} \\ & & & & & & & \ddots & \vdots \\ & & & & & & & & h_k \end{array}\right] \tag{2}$$

which has last $k-1$ rows in Hermite normal form. The goal at this stage is to perform integer row operations on T to reduce off-diagonal entries in row one.

Because the last $k-1$ rows of T are in Hermite normal form, all columns with a unit diagonal entry will have off-diagonal entries necessarily zero – row operations involving the last $k-1$ rows of T can ignore these columns. In essence, this second refinement takes advantage of the possible sparsity of the Hermite form. The following algorithm for transforming to Hermite form an input matrix as in (2) includes both refinements.

Algorithm: TriangularReduction

Input: A $k \times k$ upper triangular rank k integer matrix T with trailing $(k-1) \times (k-1)$ submatrix in Hermite normal form.

Output: The Hermite normal form of T. [T is transformed in place.]

(1) (*Initialize*:)
  $d \leftarrow \prod_{2 \leqslant i \leqslant k} T_{i,i}$;
  if $T_{1,1} < 0$ then $\text{row}(T,1) \leftarrow -\text{row}(T,1)$;
  for $j = 2$ to k do $T_{1,j} \leftarrow T_{1,j} \pmod{d}$;

(2) (*Reduce off-diagonal entries in row* 1:)
  $L \leftarrow \{j \mid 2 \leqslant j \leqslant k, T_{i,i} > 1\}$;
  for $j = 2$ to k do
  $L \leftarrow L \setminus \{j\}$;

(a) $(q,r) \leftarrow$ a solution to $T_{1,j} = qT_{j,j} + r$ with $0 \leqslant r < T_{j,j}$;
  $T_{1,j} \leftarrow r$;
  for $l \in L$ do

(b) $T_{1,l} \leftarrow T_{1,l} - qT_{j,l} \pmod{d}$;

Lemma 2. *Algorithm TriangularReduction is correct.*

Proof. It will be sufficient to show that only integer row operations are performed during the transformation and that the output matrix is in Hermite form. First note that the only entries of T which are modified during the algorithm are the off-diagonal entries of row one. Step (1) applies an integer row operation to T, if required, that ensures that entry $T_{1,1}$ is positive. By the choice of (q,r) in step (2), the output matrix will be in Hermite normal form. The fact that line (b) is essentially performing integer row operations on T follows from Lemma 1. □

Lemma 3. *If both d and $\|T\|$ are bounded by D, then the running time of Algorithm TriangularReduction is $O(k \log^2 D)$ bit operations.*

Proof. The cost of step (1) is easily seen to be bounded by $O(k \log^2 D)$ bit operations. Note that throughout step (2) all entries in the work matrix satisfy $0 \leqslant T_{i,j} \leqslant d$. A single execution of statement (a) requires $O((\lg T_{1,j}/T_{j,j})(\lg T_{j,j}))$ bit operations. Since $\lg T_{1,j}/T_{j,j} \leqslant \lg d$, we can bound this more simply by $O((\lg d)(\lg T_{j,j}))$ bit operations. The cost of a single execution of statement (b)

is bounded by $O((\lg q)(\lg T_{j,l}) + (\lg((|T_{1,l}| + |qT_{j,l}|)/d)(\lg d))$ bit operations. Replacing q, $T_{1,l}$ by $d-1$ and $T_{j,l}$ by $T_{l,l}-1$ leads to the simpler bound $O((\lg d)(\lg T_{l,l}))$ bit operations. This shows that the cost of a single execution of statement (a) and (b) is bounded by $c(\lg d)(\lg T_{j,j})$ and $c(\lg d)(\lg T_{l,l})$ respectively for some absolute constant c. The total cost for one pass of the outer loop in step (2) is given by

$$
\begin{aligned}
& c(\lg d)(\lg T_{j,j}) + \sum_{l \in L} c(\lg d)(\lg T_{l,l}) \\
& \leqslant c(\lg d)\left(\lg T_{j,j} + \sum_{l \in L} \lg T_{l,l}\right) \\
& \leqslant c(1 + \log_2 d)\left(1 + \log_2 T_{j,j} + \sum_{l \in L}(1 + \log_2 T_{l,l})\right) \\
& \leqslant c(1 + \log_2 d)\left(1 + \log_2 T_{j,j} + 2\sum_{l \in L} \log_2 T_{l,l}\right) \\
& \leqslant c(1 + \log_2 d)(1 + 2\log_2 d).
\end{aligned}
$$

Thus, one pass of the loop requires $O(\log^2 d)$ bit operations. Since the loop is repeated $k-1$ times, and $d|D$, all of step (2) can be accomplished in $O(k \log^2 D)$ bit operations. □

Theorem 4. *There exists a deterministic algorithm that receives as input an $n \times n$ rank n upper triangular matrix T and returns as output the Hermite normal form of T. If both $|\det(T)|$ and $\|T\|$ are bounded by D, then the running time of the algorithm is $O(n^2 \log^2 D)$ bit operations.*

Proof. Apply in place, for $k = 1, 2, \ldots, n$, algorithm TriangularReduction to the trailing $k \times k$ submatrix of T. □

3. Smith normal forms of triangular matrices

In this section we assume we start with an $n \times m$ integer input matrix T having principal $n \times n$ submatrix non-singular upper triangular and all entries bounded in magnitude by the product D of the diagonal entries. Eq. (3) gives an example of a 4×5 input matrix with $D = 113472$.

$$
\overset{T}{\begin{bmatrix} 3 & 113344 & 95472 & 42884 & 12302 \\ & 2 & 1576 & 98594 & 11872 \\ & & 2 & 99206 & 94692 \\ & & & 9456 & 7080 \end{bmatrix}} \rightsquigarrow \overset{S}{\begin{bmatrix} 1 & & & \\ & 2 & & \\ & & 6 & \\ & & & 24 \end{bmatrix}}. \tag{3}
$$

The goal is to transform T to Smith normal form S by applying a sequence of integer row and column operations. The classical approach for diagonalizing integer matrices using alternating row and column operations does not exploit the triangular structure during the reduction when applied to a triangular input matrix. Algorithms based on the classical approach but which perform arithmetic modulo D to avoid intermediate expression swell still require (worst case) on the order of $O(nm^2 \log D)$ arithmetic operations with integers bounded in length by $O(\log D)$ bits to compute S from T [5,6]. The near-optimal algorithm given in [11] for diagonalizing matrices over the ring of integers modulo D can be applied to the problem at hand and requires $O(nm^{\theta-1})$ arithmetic operations with integers of the same length; even assuming the current record on θ this becomes $O(nm^{1.376}(\log D)^{1+\epsilon})$ bit operations. The algorithm we propose here requires only $O(nm \log^2 D)$ bit operations using standard integer and matrix multiplication.

Before presenting the general algorithm in Section 3.3, we present two key subroutines separately in Sections 3.1 and 3.2. We will require the following simple number theoretic algorithm.

Theorem 5 (Bach [15]). *Let a, b, N be integers with N positive and $\gcd(a, b, N) = 1$. There exists a deterministic algorithm that takes as input a, b and N and returns as output an integer c with $0 \leqslant c < N$ and such that $\gcd(a + cb, N) = 1$. If $|a|, |b| \leqslant N$ then the cost of the algorithm is $O(\log^2 N)$ bit operations assuming standard integer arithmetic.*

Proof. The algorithm and proof are due to Bach [15]. The algorithm can be expressed as follows:

1. if $\gcd(a, N) = 1$ then
2. $\quad c \leftarrow 0$
3. elif $\gcd(a + b, N) = 1$ then
4. $\quad c \leftarrow 1$
5. else
6. $\quad g \leftarrow \gcd(a, N)$;
7. $\quad (N', N'') \leftarrow$ a factorization $N = N'N''$ with $\gcd(N', N'') = 1$ and $\gcd(g, N'') = 1$;
8. $\quad c \leftarrow$ an integer with $0 < c < N$, $c \equiv 1 \bmod N'$ and $c \equiv 0 \bmod N''$;
9. fi.

The values returned for c in lines 2 and 4 are clearly correct. If $\gcd(a, N) \neq 1$ then g must contain at least one prime divisor of N. If $\gcd(a + b, N) \neq 1$ as well, then some prime divisor of N must be excluded from a (otherwise $\gcd(b, N) = 1$ and we would have $\gcd(a + b, N) = 1$). The factorization $N'N''$ is obtained by applying a factor refinement algorithm to $g(N/g)$. The factor N' will contain those prime of N that are common to a while the factor N'' those primes of N that are not common to a – this reduces the problem to the

two cases in lines 2 and 4. Line 7 can accomplished in $O(\log^2 N)$ bit operations (see [13] or [16]) and line 8 is accomplished in the same time using the Chinese remainder algorithm. □

3.1. Phase one subroutines

Let T be a $k \times m$ rank k upper triangular matrix with first $k-1$ columns in Smith normal form. We can write T as

$$T = \left[\begin{array}{ccccc|ccccc} a_1 & & & & t_1 & * & * & \cdots & * \\ & a_2 & & & t_2 & * & * & \cdots & * \\ & & \ddots & & & \vdots & & & \\ & & & a_{k-1} & t_{k-1} & & & & \\ & & & & t_k & * & * & \cdots & * \end{array}\right]. \tag{4}$$

The purpose of this section is to prove the following result.

Theorem 6. *Let T be as in Eq. (4) and satisfy the conditions*

1. *first k columns of T have rank k,*
2. *first $k-1$ columns of T are in Smith normal form,*
3. *off-diagonal entries in rows $1, 2, \ldots, k-1$ are reduced modulo the diagonal entry in the same row,*
4. *off-diagonal entries in row k are bounded in magnitude by D, a positive multiple of the determinant of the principal kth submatrix of T.*

There exists a deterministic algorithm that applies integer row and column operations to transform the kth principal submatrix of T to Smith normal form. If the input matrix satisfies $k = 1$ or $a_1 > 1$, then the running time of the algorithm is $O(m \log^2 D)$ bit operations.

We first prove some intermediate results.

Lemma 7. *Let T be as in Eq. (4) with $k > 1$ and satisfy the four conditions of Theorem 6. There exists a deterministic algorithm that transforms T to an equivalent matrix that satisfies the same conditions but with the addition of*

5. $\gcd(a_i, t_i) = \gcd(a_i, t_i, t_{i+1}, \ldots, t_k)$ *for* $1 \leqslant i \leqslant k-1$.

If the input matrix satisfies $a_1 > 1$, then the running time of the algorithm is bounded by $O((m-k+1)\log^2 D)$ bit operations.

Proof. The algorithm is inductive. Note that for $r = k$ the input matrix T trivially satisfies

$$\gcd(T_{r,r}, T_{r,k}) = \gcd(T_{r,r}, T_{r,k}, T_{r+1,k}, \ldots, T_{k,k}). \tag{5}$$

For some i, $1 \leqslant i \leqslant k$, assume that T satisfies Eq. (5) for $r = k, k-1, \ldots, i+1$. We show how to apply integer row and column operations to transform T to an equivalent matrix that satisfies conditions 1–4 and Eq. (5) for $r = k, k-1, \ldots, i$. Let c be a solution to $\gcd(t_i + ct_{i+1}, a_i) = \gcd(t_i, t_{i+1}, a_i)$ with $0 \leqslant c < a_i$. Set $\mathrm{row}(T,i) = \mathrm{row}(T,i) + c\,\mathrm{row}(T,i+1)$ to produce the matrix

$$T' = \left[\begin{array}{ccccccc|cccc}
a_1 & & & & & & t_1 & * & * & \cdots & * \\
& a_2 & & & & & t_2 & * & * & \cdots & * \\
& & \ddots & & & & \vdots & \vdots & & & \\
& & & a_i & ca_{i+1} & & t_i + ct_{i+1} & * & & & \\
& & & & a_{i+1} & & t_{i+1} & * & & & \\
& & & & & \ddots & \vdots & \vdots & & \cdot & \\
& & & & & a_{k-1} & t_{k-1} & * & * & \cdots & * \\
& & & & & & t_k & * & * & \cdots & *
\end{array}\right].$$

Now,

$$\begin{aligned}
\gcd(T'_{i,i}, T'_{i,k}) &= \gcd(a_i, t_i + ct_{i+1}) \\
&= \gcd(a_i, t_i, t_{i+1}) \\
&= \gcd(a_i, t_i, a_{i+1}, t_{i+1}) \\
&= \gcd(a_i, t_i, t_{i+1}, t_{i+2}, \ldots, t_k).
\end{aligned}$$

Here, the second last equality follow from the fact that $a_i | a_{i+1}$, and the last equality follows by the induction hypothesis. Finally, reduce off-diagonal entries in row i of T' modulo the diagonal entry a_i. Since $a_i | a_{i+1}$, the entry in the ith row $(i+1)$st column will be zeroed out, leaving the first $k-1$ columns unchanged. The following code implements the above construction.

1. for $i = k-1$ by -1 to 1 do
2. $\quad c \leftarrow$ a solution to $\gcd(t_i + cT_{i+1,k}, a_i) = \gcd(t_i, T_{i+1,k}, a_i)$ with $0 \leqslant c < a_i$;
3. $\quad$ for $j = k$ to m do $T_{i,j} \leftarrow T_{i,j} + cT_{i+1,j}$;
4. $\quad$ for $j = k$ to m do $T_{i,j} \leftarrow T_{i,j} \pmod{a_i}$;

Remark 8. All the code fragments we present perform operations inplace. In the code fragment above the quantities $a_1, a_2, \ldots, a_{k-1}$ and $t_1, t_2, \ldots, t_k$ always refer to the numbers in the original input matrix as in Eq. (4). Thus, t_i is the entry in row 1 column k of the input matrix, and not necessarily the current entry in row 1 column k of the work matrix; this entry may have changed. To refer the current entry in row i column j we write $T_{i,j}$.

For convenience, we will denote D by a_k. Then the row operation at line 3 requires $O((m-k+1)(\lg a_{i+1})(\lg a_i))$ bit operations. This bounds the cost of line 2, which is accomplished using the algorithm of Lemma 5. After line

3 completes entries in row i will have magnitude bounded by $a_i + a_i(a_{i+1} - 1) = a_i a_{i+1}$ and it follows that one pass of the loop at line 4 requires $O((m-k+1)(\lg a_i a_{i+1}/a_i)(\lg a_i))$ bit operations. Thus, one pass of the outer loop at line 1 is bounded by $c(m-k+1)\lg^2 a_{i+1}$ bit operations for some absolute constant c. The total cost of the outer loop, ignoring for the moment the factor $(m-k+1)$, is given by

$$\begin{aligned}\sum_{1\leqslant i\leqslant k-1} c\lg^2 a_{i+1} &\leqslant c\left(\lg^2 a_k + \sum_{1\leqslant i\leqslant k-2}\lg^2 a_{i+1}\right)\\ &\leqslant c\left(1+\log^2 D + \sum_{1\leqslant i\leqslant k-2}(1+\log^2 a_{i+1})\right)\\ &\leqslant c\left(2\log^2 D + 2\sum_{1\leqslant i\leqslant k-2}\log^2 a_{i+1}\right)\\ &\leqslant 2c(\log^2 D + \log^2 D).\end{aligned}$$

Thus, the running time of the algorithm is bounded by $O((m-k+1)\log^2 D)$ bit operations. □

Lemma 9. *Let T be as in Eq. (4) with $k > 1$ and satisfy the five conditions of Lemma 7. There exists a deterministic algorithm that transforms T to an equivalent matrix that satisfies the same conditions but with the addition of*

6. *$T_{1,1}$ divides all other entries in the principal kth submatrix of T.*
7. *$T_{1,k} = 0$.*

If the input matrix satisfies $a_1 > 1$, then the running time of the algorithm is $O((m-k+1)(\log D)(\log a_1))$ bit operations.

Proof. We show how to transform T to the equivalent matrix

$$T' = \left[\begin{array}{ccccc|cccc} s_1 & & & & & * & * & \cdots & * \\ & a_2 & & & t_2 a_1/s_1 & * & * & \cdots & * \\ & & \ddots & & \vdots & \vdots & & & \\ & & & a_{k-1} & t_{k-1}a_1/s_1 & & & & \\ & & & & t_k a_1/s_1 & * & * & \cdots & * \end{array}\right],$$

where $s_1 = \gcd(a_i, t_i)$, using only unimodular row and column operations. Let (s,t,s_1) be a solution to the extended gcd equation $sa_1 + tt_1 = s_1$ and let V be the $m \times m$ identity matrix except with $V_{1,1} = s$, $V_{k,1} = t$, $V_{1,k} = t_1/s_1$ and $V_{k,k} = a_1/s_1$. Then V has determinant ± 1 and thus is the product of elementary matrices. Moreover,

$$TV = \left[\begin{array}{ccccc|cccc} s_1 & & & & & * & * & \cdots & * \\ tt_2 & a_2 & & & t_2a_1/s_1 & * & * & \cdots & * \\ tt_3 & & \ddots & & \vdots & \vdots & & & \\ \vdots & & & a_{k-1} & t_{k-1}a_1/s_1 & & & & \\ tt_k & & & & t_ka_1/s_1 & * & * & \cdots & * \end{array}\right],$$

with the entry in row one column k zero. By assumption, a_1 divides t_i for $i = 1,2,3,\ldots,k$. Since $s_1|a_1$, we can add appropriate multiples of row one in matrix TV to rows $2,3,\ldots,k$ to zero out off-diagonal entries in column one and produce the matrix T'. Finally, reduce off-diagonal entries in the first $k-1$ rows of T' modulo the diagonal entry in the same row, and reduce off-diagonal entries in row k modulo D. The output matrix will satisfy all seven conditions of the lemma. The following code implements the above construction. For convenience, a_k is taken to be D.

1. $(s,t,s_1) \leftarrow$ a solution to $sa_1 + tt_1 = s_1$ with $s_1 = \gcd(a_1,t_1)$;
2. $T_{1,1} \leftarrow s_1$;
3. $T_{1,k} \leftarrow 0$;
4. for $i = 2$ to k do
5. $\quad q \leftarrow tt_i/s_1 \pmod{a_i}$;
6. $\quad$ for $j = k+1$ to m do $T_{i,j} \leftarrow T_{i,j} - qT_{1,1}$;
7. $\quad T_{i,k} \leftarrow t_ia_1/s_1$;
8. $\quad$ for $j = k$ to m do $T_{i,j} \leftarrow T_{i,j} \pmod{a_i}$;
9. for $j = k$ to m do $T_{1,k} \leftarrow T_{1,k} \pmod{s_1}$.

Line 1 can be accomplished in $O((\lg a_1)^2)$ bit operations to yield a solution (s,t,s_1) satisfying $|s|,|t|,s_1 \leqslant a_1$. Since magnitudes of off-diagonal entries in rows one and i are bounded by a_1 and a_i respectively, line 6 requires $O((m-k+1)(\lg a_i)(\lg a_1))$ bit operations. This bounds the cost of lines 5 and 7 as well. After lines 5, 6 and 7 are completed, row i of T has entries bounded in magnitude by $2a_ia_1$, leading to a cost of $O((m-k+1)(\lg a_i)(\lg a_1))$ for the loop at line 8. The total cost for the loop in line 4, ignoring for the moment the factor $(m-k+1)$ is given by

$$\begin{aligned} &\sum_{i=2}^{k} c(\lg a_i)(\lg a_1) \\ &\quad = c(\lg a_k)(\lg a_1) + c(\lg a_1)\sum_{i=2}^{k-1}(\lg a_i) \\ &\quad \leqslant c(1+\log D)(1+\log a_1) + c(1+\log a_1)\sum_{i=2}^{k-1}(1+\log a_k) \\ &\quad \leqslant c(2\log D)(2\log a_1) + c(2\log a_1)\sum_{i=2}^{k-1}(2\log a_k) \\ &\quad \leqslant c(2\log D)(2\log a_1) + c(2\log a_1)(2\log D). \end{aligned}$$

Thus, the total cost of all iterations of the loop in line 4 is bounded by $O((m-k+1)(\log D)(\log a_1))$ bit operations. This bounds the cost of line 1 and the loop in line 9. □

We can now prove Theorem 6.

Proof of Theorem 6. Let T be a $k \times m$ matrix which satisfies the conditions of the Theorem. If $k > 1$, apply the algorithm of Lemma 7 at a cost of $O((m-k+1)\log^2 D)$ bit operations to "condition" T. Next, apply the algorithm of Lemma 9 to the trailing $i \times i$ submatrix of T for $i = k, k-1, \ldots, 2$ at a cost of

$$\sum_{1 \leqslant i \leqslant k} c(m-k+1)(\log D)(\log a_i) = c(m-k+1)(\log D) \sum_{1 \leqslant i \leqslant k} (\log a_i)$$
$$\leqslant c(m-k+1)(\log^2 D).$$

So far the total cost is bounded by $O((m-k+1)\log^2 D)$ bit operations. It remains to replace $T_{k,k}$ by $|T_{k,k}|$ and reduce off-diagonal entries in row k modulo $T_{k,k}$. Since $T_{k,k} \leqslant D$, the cost of this is also bounded $O((m-k+1)\log^2 D)$ bit operations. □

3.2. Phase two subroutines

Let T be a $k \times m$ rank k upper triangular matrix with first k columns in Smith normal form. We can write T as

$$T = \left[\begin{array}{cccc|ccc} a_1 & & & & b_{1,1} & \cdots & b_{1,m-k} \\ & a_2 & & & b_{2,1} & \cdots & b_{2,m-k} \\ & & \ddots & & \vdots & & \\ & & & a_k & b_{k,1} & \cdots & b_{k,m-k} \end{array}\right]. \tag{6}$$

The purpose of this section is to prove the following result.

Theorem 10. *Let T be a $k \times m$ integral matrix with first k columns rank k and in Smith normal form and off-diagonal entries in each row reduced modulo the diagonal entry in the same row. There exists a deterministic algorithm that applies integer column operations to transform T to an equivalent matrix with principal kth submatrix lower triangular and last $m-k$ columns zero. If $T_{1,1} > 1$, and the determinant of the principal kth submatrix of T is bounded in magnitude by D, then the output matrix will have all entries bounded in magnitude by D and the running time of the algorithm is $O((m-k)\log^2 D)$ bit operations.*

Proof. Let (s,t) be a solution to the extended gcd problem $sa_1 + tb_{1,j} = s_1$ where $s_1 = \gcd(a_1, b_{1,j})$. Let V be the $m \times m$ identity matrix except with

$V_{1,1} = s$, $V_{j,1} = t$, $V_{1,j} = -b_{1,j}/s_1$ and $V_{k,k} = a_1/s_1$. Then V has determinant +1 and

$$TV = \left[\begin{array}{cccc|ccc} s_1 & & & & b_{1,2} & \cdots & b_{1,m-k} \\ tb_{2,1} & a_2 & & & b_{2,1}a_1/s_1 & b_{2,2} & \cdots & b_{2,m-k} \\ \vdots & & \ddots & & \vdots & & & \\ tb_{k,1} & & & a_k & b_{k,1}a_1/s_1 & b_{k,2} & \cdots & b_{k,m-k} \end{array}\right],$$

with the entry in row one column $k+1$ zero. Now, reduce off-diagonal entries in columns 1 and j in TV modulo the diagonal entries in the same row. Repeating this process, zero out all off-diagonal entries in row 1. The following code implements the above construction.

1. for $j = 1$ to $m-k$ do
2. $\quad (s,t,s_1) \leftarrow$ a solution to $sT_{1,1} + tb_{1,j} = s_1$ with $s_1 = \gcd(T_{1,1}, b_{1,j})$;
3. $\quad a \leftarrow -b_{1,j}/s_1$;
4. $\quad b \leftarrow T_{1,1}/s_1$;
5. $\quad T_{1,1} \leftarrow s_1$;
6. $\quad T_{1,j+k} \leftarrow 0$;
7. $\quad$ for $i = 2$ to k do
8. $\quad\quad C \leftarrow sT_{i,1} + tb_{i,j} \pmod{a_i}$;
9. $\quad\quad T_{i,j+k} \leftarrow aT_{i,1} + bb_{i,j} \pmod{a_i}$;
10. $\quad\quad T_{i,1} \leftarrow C$.

Since all entries in row i of T are reduced modulo a_i, the quantities s_1, s, t, a and b computed in lines 2, 3 and 4 will have magnitude bounded by a_1 and the cost of one pass of these lines is bounded by $O((\lg^2 a_1))$ bit operations. The cost of a single pass of the loop at line 7 is $O((\lg a_1)(\lg a_i))$ bit operations. The total cost of the loop at line 7 for a single value of j is given by

$$\begin{aligned} \sum_{2 \le i \le k} c(\lg a_1)(\lg a_i) &= c(1 + \log a_1) \sum_{2 \le i \le i} (1 + \log a_i) \\ &\le c(2\log a_1) \sum_{2 \le i \le k} (2\log a_i) \\ &\le c(2\log a_1)(2\log D). \end{aligned}$$

Thus, the total cost of the loop at line 7 for a single value of the outer loop index j is bounded by $O((\log a_1)(\log D))$ bit operations, and this bounds the cost of lines 2, 3 and 4 as well. The outer loop at line 1 is repeated $m-k$ times so the total cost of the code fragment is $O((m-k)(\log a_1)(\log D))$ bit operations.

After the code completes the work matrix has the form

$$\left[\begin{array}{cccc|ccccc} s_1 & & & & & & & & \\ * & a_2 & & & * & * & \cdots & * & \\ \vdots & & \ddots & & \vdots & & & & \\ * & & & a_k & * & * & \cdots & * & \end{array}\right],$$

with all off-diagonal entries in row one zero. To complete the reduction, apply a similar procedure to the trailing $(k-i+1) \times (m-i+1)$ submatrix of the work matrix to zero out entries to the right of the diagonal in row i for $i = 2, 3, \ldots, k$. The total cost to reduce T to lower triangular form is

$$\sum_{1 \leqslant i \leqslant k} c(m-k)(\log a_i)(\log D) \leqslant c(m-k) \log^2 D$$

bit operations for some absolute constant c. □

3.3. The triangular Smith normal form algorithm

Theorem 11. *There exists a deterministic algorithm that receives as input an $n \times m$ integral matrix A with principal $n \times n$ submatrix non-singular and upper triangular, and returns the Smith normal form of A. If both the magnitude of all entries of A and the magnitude of the determinant of the principal $n \times n$ submatrix of A is bounded by D, then the running time of the algorithm is $O(nm \log^2 D)$ bit operations using standard integer arithmetic.*

Proof. Phase one of the algorithm transforms, for $r = 1, 2, \ldots, n$ in succession, the principal $r \times r$ submatrix of A into Smith form. The algorithm is inductive and it is sufficient to consider a single stage. Let $A^{(r)}$ be the work matrix at the beginning of stage r. Off-diagonal entries in rows $1, 2, \ldots, r-1$ of $A^{(r)}$ will have magnitude bounded by the diagonal entry in the same row, and using a block decomposition, the $n \times m$ matrix $A^{(r)}$ can be written as

$$A^{(r)} = \left[\begin{array}{c|c|c} I_{r-k} & & \\ \hline & T_1 & T_2 \\ \hline & & B \end{array}\right]. \tag{7}$$

T_1 is $k \times k$ upper triangular with rank k and with first $k-1$ columns in Smith form. If $k > 1$ then the leading diagonal entry of T_1 is greater than 1. T_2 is $k \times (m-r)$ with entries in rows i, $1 \leqslant i \leqslant k-1$, reduced modulo the ith diagonal entry in T_1 and entries in row k reduced modulo D. B is the trailing $(n-r) \times (m-r)$ submatrix of the input matrix A. For the initial case $r = 1$ of the induction, note that $A^{(1)}$, which is A, can be written as in Eq. (7) with $k = 1$.

At stage r, row operations on $A^{(r)}$ are limited to rows occupied by T_1 and the only column operations involving columns occupied by B are limited to those which add multiples of columns occupied by T_1 to these last $m-r$ columns. Hence, the last $n-r$ rows of the work matrix (those occupied by B) remain unchanged during stage r. Furthermore, since the principal $(r-k)$th submatrix of $A^{(r)}$ is the identity, we can limit our attention the submatrix of $A^{(r)}$ comprised of T_1 and T_2, namely the $k \times (m-r)$ matrix

$$T = [T_1|T_2] = \left[\begin{array}{ccccc|cccc} a_1 & & & & t_1 & * & * & \cdots & * \\ & a_2 & & & t_2 & * & * & \cdots & * \\ & & \ddots & & \vdots & * & * & \cdots & * \\ & & & a_{k-1} & t_{k-1} & \vdots & & & \\ & & & & t_k & * & * & \cdots & * \end{array}\right].$$

Use the algorithm of Theorem 6 to transform T to an equivalent matrix with principal $k \times k$ submatrix in Smith form and off-diagonals in each row reduced modulo the diagonal entry in each column. By Theorem 6, the cost of one stage is $O(m \log^2 D)$ bit operations. This stage is repeated for $r = 1, 2, \ldots, n$, leading to a total cost for phase one of $O(nm \log^2 D)$ bit operations.

After phase one, the work matrix either has principal nth submatrix the identity and all other entries zero (in which case we are finished) or has the form

$$\left[\begin{array}{c|c|c} I_{n-k} & & \\ \hline & S_1 & B_1 \end{array}\right],$$

where S_1 is a $k \times k$ matrix in Smith form with k chosen so that the leading diagonal entry of S_1 is greater than one. Furthermore, off-diagonal entries in each row (i.e. the entries in B_1) are reduced modulo the diagonal entry in the same row. Use the algorithm Theorem 10 to transform the submatrix $[S_1 \mid B_1]$ to an equivalent matrix with first k columns lower triangular and all other columns zero. By Theorem 10, the cost of the transformation is $O(m \log^2 D)$ bit operations. After phase two the work matrix has the form

$$\left[\begin{array}{c|c|c} I_{n-k} & & \\ \hline & A_1 & O \end{array}\right],$$

where A_1 is a $k \times k$ lower triangular with magnitude of determinant and all entries bounded by D. Finally, use the reduction of phase one to reduce the square non-singular matrix A_1 to Smith form at a cost of $O(n^2 \log^2 D)$ bit operations. Combining these phases, the total cost for the reduction is seen to be bounded by $O(nm \log^2 D)$ bit operations. □

4. An algorithm for the Smith normal form

Let A be an $n \times m$ input matrix. The first step in computing the Smith normal form is to transform A to a row equivalent matrix H which satisfies the following conditions:

(e1) Let r be the rank of H. Then the first r rows of H are non-zero.

(e2) For $1 \leqslant i \leqslant r$ let $H[i, j_i]$ be the first non-zero entry in row i. Then $j_1 < j_2 < \cdots < j_r$.

A matrix H which satisfies (e1) and (e2) and is row equivalent to A is said to be a row echelon form of A. In [17] we give a fast, practical and deterministic algorithm for recovering r and a row echelon form H of A in $O(nmr^2 \log^2 r\|A\| + r^4 \log^3 r\|A\|)$ bit operations. Furthermore, both $\|H\|$ and $D = |\prod_{i=1}^{r} H[i, j_i]|$ will be bounded in length by $O(r \log r\|A\|)$ bits. Let P be a permutation matrix such that column i of HP is column j_i of H for $1 \leqslant i \leqslant r$. The Smith normal form of HP (which is equivalent to A) can be computed in $O(rm \log^2 D)$ bit operations by applying the algorithm of Theorem 11 to the submatrix comprised of the first r rows of HP. Noting that $O(rm \log^2 D)$ is bounded by $O(nmr^2 \log^2 r\|A\|)$ yields the following result.

Theorem 12. *There exists a deterministic algorithm that takes as input an $n \times m$ integer input matrix A and recovers the rank r and Smith normal form S of A. The cost of the algorithm is $O(nmr^2 \log^2 r\|A\| + r^4 \log^3 r\|A\|)$ bit operations assuming standard integer and matrix arithmetic.*

Proof. Follows from Theorem 11 and [17] (Theorem 16). □

5. Conclusions and open problems

By combining the triangular Smith form algorithm presented here with the triangularization algorithm in [17] we were able to get an improved practical algorithm for computing Smith forms of a dense input matrix with arbitrary shape and rank profile. This Smith form algorithm seems to be inherently sequential, and like previous deterministic algorithms [5,6,11], requires about a factor of $O(n)$ more space than is required to write down the input or output matrix. An important advantage of Giesbrecht's [10,18] Monte Carlo probabilistic Smith normal form algorithms, in addition to allowing a simple coarse grain parallelization, is an improved space complexity.

To compare the various algorithms, consider the case of an $n \times n$ input matrix A with small entries, that is, with $\log \|A\| < c$ for some fixed constant c independent of n. Previous deterministic algorithms, including the algorithm we have presented here, require on the order of $O^{\sim}(n^3)$ additional bits of storage

space to compute the Smith form S of A. The algorithm in [10] requires only about $\tilde{O}(n^2)$ additional bits of storage.

Now consider the case when A is sparse, with on the order of $O(n \log n)$ non-zero entries. The algorithm for sparse matrices in [18] also requires only about $\tilde{O}(n^2)$ additional bits of storage. Moreover, only about $\tilde{O}(n^3)$ bit operations are required to recover S; a running time which is factor of about $\tilde{O}(n)$ faster than the algorithm we have presented here for dense matrices. This is a significant breakthrough since large sparse input matrices with small entries arise very often in practice.

The disadvantage of the algorithms in [10,18] is that they require randomization and return a Smith form which may, with controllable, exponentially small probability of error, be incorrect. An important open problem is to find a fast algorithm for computing Smith normal forms of sparse integer input matrices which guarantees correctness and also admits near-optimal space complexity. Since finding the Smith normal form also recovers the rank of the input matrix, a simpler open problem to consider is a Las Vegas probabilistic algorithm that requires $\tilde{O}(n^3)$ bit operations using standard arithmetic to recover only the rank of A. The currently fastest rank algorithm which guarantees correctness requires $O(n^{\theta+1})$ bit operations.

References

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

[2] A. Schönhage, V. Strassen, Schnelle multiplikation grosser zahlen, Computing 7 (1971) 281–292.

[3] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, J. Symbolic Comput. 9 (1990) 251–280.

[4] P.D. Domich, R. Kannan, L.E. Trotter, Jr., Hermite normal form computation using modulo determinant arithmetic, Math. Oper. Res. 12 (1) (1987) 50–59.

[5] J.L. Hafner, K.S. McCurley, Asymptotically fast triangularization of matrices over rings, SIAM J. Comput. 20 (6) (1991) 1068–1083.

[6] C.S. Iliopoulos, 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 J. Comput. 18 (4) (1989) 658–669.

[7] A. Storjohann, G. Labahn, Asymptotically fast computation of Hermite normal forms of integer matrices, in: Y.N. Lakshman (Ed.), Proceedings of the International Symposium on Symbolic and Algebraic Computation: ISSAC'96, ACM (Assoc. Comput. Machinery), New York, 1996, pp. 259–266.

[8] H.J.S. Smith, On systems of linear indeterminate equations and congruences, Philos. Trans. Roy. Soc. London 151 (1861) 293–326.

[9] M. Newman, The Smith normal form, Linear Algebra Appl. 254 (1997) 367–381.

[10] M. Giesbrecht, Fast computation of the Smith normal form of an integer matrix, in: A.H.M. Levelt (Ed.), Proceedings of the International Symposium on Symbolic and Algebraic Computation: ISSAC'95, 1995, pp. 110–118.

[11] A. Storjohann, Near optimal algorithms for computing Smith normal forms of integer matrices, in: Y.N. Lakshman (Ed.), Proceedings of the International Symposium on Symbolic and Algebraic Computation: ISSAC'96, ACM (Assoc. Comput. Machinery), New York, 1996, pp. 267–274.
[12] G.E. Collins, Computing time analyses for some arithmetic and algebraic algorithms, Technical Report 36, University of Wisconsin, Madison; Computer Sciences, July 1968; in: Proceedings of 1968 Summer Institute on Symbolic Mathematical Computations, IBM Federal Systems Center, 1968, pp. 195–231.
[13] E. Bach, J. Shallit, Algorithmic Number Theory, vol. 1: Efficient Algorithms, MIT Press, Cambridge, MA, 1996.
[14] T.-W.J. Chou, G.E. Collins, Algorithms for the solutions of systems of linear diophantine equations, SIAM J. Comput. 11 (1982) 687–708.
[15] E. Bach, Linear algebra modulo N, unpublished manuscript, December 1992.
[16] E. Bach, J. Driscoll, J.O. Shallit, Factor refinement, J. Algorithms 15 (1993) 199–222.
[17] A. Storjohann, A fast, practical and deterministic algorithm for triangularizing integer matrices, Technical Report 255, Departement Informatik, ETH Zürich, December 1996.
[18] M. Giesbrecht, Probabalistic computation of the Smith normal form of a sparse integer matrix, in: H. Cohen (Ed.), Algorithmic Number Theory: Second International Symposium, 1996, pp. 175–188.