A Cubic Algorithm for Computing the Hermite Normal Form of a Nonsingular Integer Matrix

STAVROS BIRMPILIS, GEORGE LABAHN, and ARNE STORJOHANN, University of Waterloo, Canada

A Las Vegas randomized algorithm is given to compute the Hermite normal form of a nonsingular integer matrix A of dimension n. The algorithm uses quadratic integer multiplication and cubic matrix multiplication and has running time bounded by $O(n^3(\log n + \log ||A||)^2(\log n)^2)$ bit operations, where $||A|| = \max_{ij} |A_{ij}|$ denotes the largest entry of A in absolute value. A variant of the algorithm that uses pseudo-linear integer multiplication is given that has running time $(n^3 \log ||A||)^{1+o(1)}$ bit operations, where the exponent "$+o(1)$" captures additional factors $c_1(\log n)^{c_2}(\operatorname{loglog} ||A||)^{c_3}$ for positive real constants c_1, c_2, c_3.

CCS Concepts: • **Theory of computation → Design and analysis of algorithms**; • **Computing methodologies → Linear algebra algorithms**.

Additional Key Words and Phrases: Hermite normal form, Howell normal form, Smith massager, integer matrix

1 INTRODUCTION

Corresponding to any nonsingular integer matrix $A \in \mathbb{Z}^{n\times n}$, there is a unimodular matrix $U \in \mathbb{Z}^{n\times n}$ such that

$$H = UA = \begin{bmatrix} h_1 & h_{12} & \cdots & h_{1n} \\ & h_2 & \cdots & h_{2n} \\ & & \ddots & \vdots \\ & & & h_n \end{bmatrix}$$

has all entries nonnegative, and off-diagonal entries h_{*j} strictly smaller than the diagonal entry h_j in the same column. H is the (integer) Hermite normal form of A. The form is unique with its existence dating back to Hermite [1851]. The rows of H give a canonical basis for the lattice generated by the $\mathbb{Z}$-linear combinations of the rows of A. In addition to being upper triangular and canonical, an important property of the basis given by the Hermite form is that it requires only $O(n^2(\log n + \log ||A||))$ bits to represent, compared to $O(n^2 \log ||A||)$ to write down the input matrix.

Applications of the Hermite form are well known including, for example, solving systems of linear diophantine equations [Chou and Collins 1982], integer programming [Schrijver 1998], and determining rational invariants and rewriting rules of scaling invariants [Hubert and Labahn 2013], to name just a few.

Algorithms for computing Hermite normal forms for integer matrices were initially based on triangularizing the input matrix using variations of Gaussian elimination that used the extended Euclidean algorithm to eliminate entries below the diagonal. However, such methods can be prone to exponential expression swell, that is, the problem of rapid growth of intermediate integer operands. The first provably polynomial time algorithm was given by Kannan and Bachem [1979], with Chou and Collins [1982] improving this to a running time of $(n^6 \log ||A||)^{1+o(1)}$ bit operations. Domich et al. [1987], Iliopoulos [1989] and Hafner and McCurley [1989] later improved these to $(n^4 \log ||A||)^{1+o(1)}$. Further improvements came from Storjohann and Labahn [1996] and Storjohann [2000], with worst case time complexity bounded by $(n^{\omega+1} \log ||A||)^{1+o(1)}$ bit operations, where ω is the exponent of matrix multiplication. The standard algorithm for matrix multiplication has

Authors' address: Stavros Birmpilis, sbirmpil@uwaterloo.ca; George Labahn, glabahn@uwaterloo.ca; Arne Storjohann, astorjoh@uwaterloo.ca, University of Waterloo, Cheriton School of Computer Science, 200 University Ave W, Waterloo, N2L 3G1, Canada.

$\omega = 3$, while the current best known asymptotic upper bound for ω by Alman and Williams [2021] allows $\omega < 2.37286$.

Recently, a number of approaches have focused on heuristic methods to achieve faster computation, for example [Micciancio and Warinschi 2001; Pernet and Stein 2010; Pauderis and Storjohann 2013; Liu and Pan 2019] with the last citation having a complexity of $(n^{\omega} \log ||A||)^{1+o(1)}$ in the case of random input matrices. However, these algorithms require strong assumptions, for example, that there be only a small number of non-trivial ($\neq 1$) late diagonal entries of the Hermite form, something common with random matrices.

In this paper, we give a new randomized algorithm for computing the Hermite normal form of a nonsingular integer matrix $A \in \mathbb{Z}^{n \times n}$. Assuming the use of standard (quadratic) integer multiplication and standard (cubic) matrix multiplication, the algorithm has a worst case running time bounded by $O(n^3(\log n + \log ||A||)^2(\log n)^2)$ bit operations. If we use a subcubic matrix multiplication algorithm, for example Strassen's algorithm, then the cost is $O(n^3(\log n + \log ||A||)^2)$. We also give a variant of our algorithm that has a complexity of $(n^3 \log ||A||)^{1+o(1)}$ bit operations, assuming fast (pseudo-linear) integer multiplication. In all cases, our Hermite form algorithms are probabilistic of type Las Vegas. That is, the algorithm can report Fail with probability at most 1/2 but otherwise returns an answer that is certified to be correct. The three key ideas that we use are minimal matrix denominators, Smith massagers and duality of row Hermite and column Howell forms.

We remark that one can also define the Hermite form for a matrix of univariate polynomials with coefficients from a field. In this case, the definition requires that the diagonal elements are monic, while the off-diagonal entries have lower degree than the diagonal entry in the same column. The algorithms mentioned in the third paragraph of this section all have corresponding versions which work for the polynomial Hermite form, and have a complexity similar to the integer based algorithms, but with degree taking the place of bitlength and counting field operations instead of bit operations. However, there are new, very efficient algorithms which work in the polynomial case but which have no counterpart in the integer case. In particular we mention the recent fast algorithm of Labahn et al. [2017]. This algorithm is deterministic and computes the (polynomial) Hermite form with a complexity of $(n^{\omega} \lceil s \rceil)^{1+o(1)}$ field operations, with s being the minimum of the average of the degrees of the columns of A and that of its rows. Unfortunately, some of the tools used in that algorithm do not have counterparts in the case of integer matrices. In particular, for polynomial matrices one has notions such as degree shifts Beckermann et al. [1999], order bases Beckermann and Labahn [1994]; Zhou and Labahn [2012], column bases Zhou and Labahn [2013] and minimal nullspace bases Zhou et al. [2012] along with algorithms for their fast computation. For example, the fast Hermite algorithm of Labahn et al. [2017] works by directly triangularizing the input matrix, but is able to exploit the aforementioned tools, that are particular to polynomial matrices, in order keep degrees of intermediate polynomials controlled while at the same time maintaining a good complexity.

The rest of this paper is organized as follows. Section 2 gives an overview of our approach. Sections 3 and 4 introduce the mathematical and computational tools we use, including minimal denominators, Smith massagers, compact representations of both Hermite forms and Smith massagers, and some basic subroutines. Section 5 then gives an algorithm for determining the diagonal elements of the Hermite form. Section 6 describes the column Howell form of a matrix over $\mathbb{Z}/(s)$ for positive modulus s, while Section 7 relates the column Howell form to the inverse of the Hermite form. Section 8 then shows how we compute the Hermite form from a Howell form corresponding to the inverse of the Hermite form, with Section 9 detailing our modification of Howell's algorithm to compute a transformation matrix to produce the required Howell form.

Section 10 gives an algorithm to compute a type of scaled matrix vector product which is essential to obtaining the running time bound of our algorithm. Section 11 uses the results of the previous section to obtain our main result: a Las Vegas algorithm for the Hermite form with expected running time $O(n^3(\log n + \log ||A||))^2(\log n)^2)$ bit operations assuming standard integer and matrix multiplication. Section 12 gives a variant of the algorithm that has running time $(n^3 \log ||A||)^{1+o(1)}$ bit operations assuming fast (pseudo-linear) integer multiplication. The final section gives a conclusion along with some topics for future research.

Cost 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 integer arithmetic, a and b can be multiplied in $O((\lg a)(\lg b))$ bit operations, and we can express $a = qb + r$, with $0 \leq |r| < |b|$, in $O((\lg a/b)(\lg b))$ bit operations. This complexity model was popularized by Collins [1968] and is sometimes called "naive bit complexity" (see, for example, Bach and Shallit [1996]).

For an integer vector v, it will be convenient to define the *bitlength of* v to mean the bitlength of the largest entry of v in absolute value.

2 OUR APPROACH

In this section, we give a high level description of our approach for computing the Hermite form $H \in \mathbb{Z}^{n\times n}$ of a nonsingular input matrix $A \in \mathbb{Z}^{n\times n}$. As previously mentioned, there is a unimodular matrix $U \in \mathbb{Z}^{n\times n}$ such that $H = UA$. Multiplying both sides of this equation on the right by A^{-1} gives

$$HA^{-1} = U. \tag{1}$$

The basis of our approach is to recast the problem of computing H, a unimodular row triangularization of A, into that of finding a *minimal left denominator* of A^{-1}. It follows from the uniqueness of the Hermite form that H can be defined to be the matrix (in Hermite form) that clears the denominators of A^{-1} under premultiplication and has minimal determinant (i.e., $\det H = |\det A|$, since $\det U = \pm 1$).

To avoid working with fractions, define $A^* = sA^{-1}$, where $s \in \mathbb{Z}_{>0}$ is minimal such that sA^{-1} is integral. Then $HA^{-1} \in \mathbb{Z}^{n\times n}$ holds if and only if

$$HA^* = 0_{n\times n} \bmod s.$$

Unfortunately, A^* requires $\Omega(n^3(\log n + \log ||A||))$ bits to write down in the worst case, and by working with A^* explicitly we do not know how to achieve our target complexity. However, this approach allows us to bring the Smith form of A into play and reduce the space requirements.

Let $S = \mathrm{diag}(s_1, \ldots, s_n =: s)$ be the Smith form of A, and let $V, W \in \mathbb{Z}^{n\times n}$ be unimodular matrices satisfying $AV = WS$. Then,

$$A^* \equiv_R VS^*$$

where $S^* = sS^{-1} \in \mathbb{Z}^{n\times n}$ and $\equiv_R$ denotes right equivalence by unimodular matrices over $\mathbb{Z}$. Such an equivalence also holds modulo s for a matrix $M = \mathbf{c}\mathrm{mod}(V, S)$. Here, $\mathbf{c}\mathrm{mod}$ denotes working modulo columns: column j of M is equal to column j of V reduced modulo s_j, $1 \leq j \leq n$. The matrix M is called a *reduced Smith massager* of A. The fact that

$$A^* \equiv_R MS^* \bmod s \tag{2}$$

then implies that A^{-1} and MS^{-1} have the same minimal left denominator in Hermite form, namely, for any $H \in \mathbb{Z}^{n\times n}$, we have $HA^* = 0_{n\times n} \bmod s$ if and only if

$$HMS^* = 0_{n\times n} \bmod s.$$

This allows us to look for a minimal left denominator in Hermite form for a matrix with total size controlled by the Smith form S: the space required to store M is $O(n^2(\log n + \log ||A||))$ bits. Moreover, there is an existing algorithm that can compute both S and M quickly.

The special form of the matrix MS^{-1} and the uniqueness of Hermite forms has a number of advantages for efficient computation. First, by using an algorithm of Pauderis and Storjohann [2013], we can find a minimal triangular denominator for MS^{-1}, expressed as a product of n minimal Hermite denominators. While this does not produce the Hermite form H of A, the product of the diagonals of these n triangular matrices gives the diagonal entries of H. We show that the overall cost of obtaining the diagonal entries of H from M and S is $O(n(\log \det S)^2)$ bit operations. This allows us to overcome one of the biggest issues in designing a fast algorithm for the Hermite form in the worst case, that is, we now know the bitlength of each of the columns of H.

Notice that finding H^{-1} is equivalent to finding the Hermite form, since H is triangular. Indeed, let H_j be equal to I_n except with column j equal to that of H, $1 \le j \le n$. Then, since both H and H^{-1} are upper triangular, there is a simple iterative scheme to go from H^{-1} to H shown in Figure 1. We remark that in the first line of the j-loop in Figure 1, the principal leading $(j-1)\times(j-1)$ submatrix of $\bar{H}$ will be I_{j-1}, and column j of $\bar{H}$ will have the form

$$-\frac{1}{h_j}\begin{bmatrix} h_{1j} \\ \vdots \\ h_{j-1,j} \\ -1 \\ \\ \\ \end{bmatrix},$$

from which H_j is easily recovered.

$\bar{H} := H^{-1}$
for $j = 1$ **to** n **do**
  Recover H_j from column j of $\bar{H}$
  $\bar{H} := H_j\bar{H}$
od
return $H_nH_{n-1}\cdots H_1$

Fig. 1. Hermite form H from H^{-1}

However, we can do better. The same process can work without having H^{-1} exactly. Since there exists a unimodular matrix U such that $UA = H$, then by letting $H^* = sH^{-1}$, we can write this as the dual problem

$$A^*U^* = H^*$$

with U^* unimodular. Since H^* is an upper triangular integer matrix, we later show that we can replace H^* by any upper triangular matrix having the same diagonal entries and which is right equivalent to H^* modulo s. The natural form for such a matrix is the column Howell form T, a type of column reduced echelon matrix over the residue class ring $\mathbb{Z}/(s)$.

This implies that we can construct the Hermite form from any column Howell form T that is right equivalent to H^* over $\mathbb{Z}/(s)$. This allows us to replace H^{-1} by T in the procedure shown in Figure 1, and to work modulo s, and thus avoid explicit fractions.

Example 1. *Let*

$$A = \begin{bmatrix} -13 & 10 & -20 & 27 \\ 27 & 30 & 15 & 30 \\ 0 & 15 & 15 & 6 \\ -21 & 0 & -15 & 9 \end{bmatrix}.$$

[Birmpilis et al. 2020, Algorithm `SmithMassager`*] gives the Smith form* $S = \mathrm{diag}(s_1, s_2, s_3, s_4 =: s) = \mathrm{diag}(1, 3, 15, 105 =: s)$ *and a Smith massager* M *for* A *as*

$$M := \begin{bmatrix} 0 & 2 & 0 & 55 \\ 0 & 0 & 7 & 32 \\ 0 & 2 & 2 & 41 \\ 0 & 2 & 10 & 10 \end{bmatrix}.$$

Let $S^* = sS^{-1}$. *By computing a minimal denominator of* M *that is expressed as the product of four upper triangular matrices, we determine the diagonal elements of* H *to be* $h_1, h_2, h_3, h_4 = 1, 15, 15, 21$. *A Howell form of* $MS^* \in \mathbb{Z}/(s)^{n\times n}$ *with the appropriate diagonal elements of* H^* *is then given by*

$$T = \begin{bmatrix} 105 & 70 & 70 & 45 \\ & 7 & 0 & 100 \\ & & 7 & 101 \\ & & & 5 \end{bmatrix} = \begin{bmatrix} \frac{s}{h_1} & 70 & 70 & 45 \\ & \frac{s}{h_2} & 7 & 100 \\ & & \frac{s}{h_3} & 101 \\ & & & \frac{s}{h_4} \end{bmatrix}.$$

Section 7 shows that column j *of* $(H_{j-1} \cdots H_1)T$ *is congruent modulo* s *to*

$$-\frac{s}{h_j} \begin{bmatrix} h_{1j} \\ \vdots \\ h_{j-1,j} \\ -1 \\ \\ \\ \end{bmatrix} \bmod s,$$

from which H_j is easily recovered. Using T instead of H^{-1} in the procedure of Figure 1 and working modulo 105 then gives

$$j = 1: \qquad H_1 = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \text{ and } H_1 T = T$$

$$j = 2: \qquad H_2 = \begin{bmatrix} 1 & 5 & & \\ & 15 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \text{ and } H_2 H_1 T = \begin{bmatrix} & & 70 & 20 \\ & & 0 & 30 \\ & & 7 & 101 \\ & & & 5 \end{bmatrix}$$

$$j = 3: \qquad H_3 = \begin{bmatrix} 1 & & 5 & \\ & 1 & 0 & \\ & & 15 & \\ & & & 1 \end{bmatrix} \text{ and } H_3 H_2 H_1 T = \begin{bmatrix} & & & 0 \\ & & & 30 \\ & & & 45 \\ & & & 5 \end{bmatrix}$$

$$j = 4: \qquad H_4 = \begin{bmatrix} 1 & & & 0 \\ & 1 & & 15 \\ & & 1 & 12 \\ & & & 21 \end{bmatrix} \text{ and } H_4 H_3 H_2 H_1 T = 0_{4\times 4}$$

with the Hermite basis given by

$$H_4 H_3 H_2 H_1 = \begin{bmatrix} 1 & 5 & 5 & 0 \\ & 15 & 0 & 15 \\ & & 15 & 12 \\ & & & 21 \end{bmatrix}.$$

Unfortunately, as mentioned previously for A^*, the size of a Howell form T can be $\Omega(n^3(\log n + \log ||A||))$ bits in the worst case, and by working with T directly we do not know how to achieve our target complexity. Instead, we compute a matrix $\tilde{U}$ satisfying

$$T = MS^*\tilde{U} \bmod s, \tag{3}$$

where $S^* = sS^{-1}$. Furthermore, in the same way that we could assume that M was column reduced modulo S, we may assume that $\tilde{U}$ is row reduced modulo S. The number of bits required to represent all three matrices on the right hand side of (3) is then $O(n^2(\log n + \log ||A||))$.

The matrix $\tilde{U}$ can be found by a simple modification of Howell's original algorithm for determining his normal form. In order to then find column j of $H_{j-1}H_{j-2}\cdots H_1 T$, we need to determine

$$(v_1, \ldots, v_n) = (-h_{1j}, \ldots, -h_{j-1,j}, 1, 0, \ldots, 0)$$

satisfying the equation

$$\frac{s}{h_j} \underbrace{\begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}}_{v} \equiv \underbrace{\begin{bmatrix} m_{11} & \cdots & m_{1n} \\ \vdots & \ddots & \vdots \\ m_{n1} & \cdots & m_{nn} \end{bmatrix}}_{\tilde{M}} \underbrace{\begin{bmatrix} \frac{s}{s_1} & & \\ & \ddots & \\ & & \frac{s}{s_n} \end{bmatrix}}_{S^*} \underbrace{\begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}}_{u} \bmod s,$$

where $\tilde{M} = \mathrm{cmod}(H_{j-1}H_{j-2}\cdots H_1 M, S)$, and u is column j of $\tilde{U}$. To compute this matrix vector product with the intermediate scaling matrix S^*, we take advantage of the fact that $\tilde{M}$ and u are column and row reduced modulo S, respectively. We also exploit the fact that we have precomputed the diagonal entries of the Hermite form, and thus know the scaling factor s/h_j. This allows us to achieve a cost estimate for computing column j that depends on $\log ||v|| \leq \log h_j$ instead of $\log s$.

Ultimately, our algorithm computes the Hermite form in a column by column basis, with the computation for column j requiring

$$O(n^2(\log n + \log ||A||)(\log h_j + \log n + \log ||A||))$$

bit operations. Adding over all iterations $1 \leq j \leq n$ then gives the total cost of our algorithm.

3 MATHEMATICAL PRELIMINARIES

In this section, we discuss some basic mathematical building blocks used in our Hermite form algorithm. These include minimal denominators of rational matrices, Smith massagers of A, and data structures for the compact representation of Hermite forms and Smith massagers.

3.1 Minimal denominators

Definition 2. *A* (left) denominator *of a matrix $B \in \mathbb{Q}^{n\times m}$ is a matrix $H \in \mathbb{Z}^{n\times n}$ whose rows are in the lattice*

$$\{v \in \mathbb{Z}^{1\times n} \mid vB \in \mathbb{Z}^{1\times m}\}. \tag{4}$$

H is a minimal denominator *if the rows of H are a basis for (4). The* minimal Hermite denominator *is the unique minimal denominator that is in Hermite form.*

For example, a minimal denominator of a zero matrix with n rows is I_n, while a minimal denominator of A^{-1} is A itself. The minimal Hermite denominator of A^{-1} is H, the Hermite form of A. Similarly, if A^{-1} and B^{-1} are right equivalent then they have the same minimal Hermite denominator.

Example 3. *The minimal Hermite denominator of*

$$\frac{1}{16}\begin{bmatrix} 1 \\ 4 \\ 4 \\ 8 \end{bmatrix} \in \mathbb{Q}^{4\times 1}$$

is

$$H = \begin{bmatrix} 4 & 0 & 1 & 1 \\ & 1 & 1 & 1 \\ & & 2 & 1 \\ & & & 2 \end{bmatrix} \in \mathbb{Z}^{4\times 4}.$$

This shows that a rational matrix with n rows but with fewer than n columns can encode a nontrivial $n \times n$ Hermite form.

The next two lemmas follow from the fact that a minimal denominator is a basis for the lattice shown in (4).

Lemma 4. *Any two minimal denominators for a $B \in \mathbb{Q}^{n\times m}$ are left equivalent over $\mathbb{Z}$.*

Lemma 5. *The determinant of a minimal denominator for a $B \in \mathbb{Q}^{n\times m}$ divides the determinant of any other denominator of B.*

Important for our work is that minimal denominators can be computed in parts as shown by the following lemma.

Lemma 6. *Decompose $B \in \mathbb{Q}^{n\times m}$ arbitrarily as $B = \begin{bmatrix} B_1 & | & B_2 \end{bmatrix}$. If H_1 is a minimal denominator of B_1, and H_2 is a minimal denominator of H_1B_2, then H_2H_1 is a minimal denominator of B.*

Proof. It is evident that H_2H_1 is a denominator of B, and hence, we only need to show that it is minimal. If it is not a minimal denominator, then there exist matrices $H, W \in \mathbb{Z}^{n\times n}$ such that H is a minimal denominator, $H_2H_1 = WH$ and W is not unimodular.

However, since $H_2 = WHH_1^{-1}$ is a minimal denominator of H_1B_2, then WH must be a minimal denominator of B_2. This is a contradiction since H is a denominator of B_2 and W is not unimodular. □

Finally, recall that any rational number can be written as an integer and a proper fraction. For example,

$$\frac{9622976468279041913}{21341} = 450914974381661 + \frac{14512}{21341}, \tag{5}$$

where 450914974381661 is the quotient and 14512 is the remainder of the numerator with respect to the denominator. We see that, for any rational matrix B, if s is a positive integer such that sB is integral, then the proper fraction $\mathrm{Rem}(sB, s)/s$ and B have the same denominators. Here, Rem denotes the positive remainder. Thus, instead of working with the rational matrix B, we can work with the matrix $\mathrm{Rem}(sB, s)$ over $\mathbb{Z}/(s) = \{0, 1, \ldots, s-1\}$.

Lemma 7. *For $B \in \mathbb{Q}^{n\times m}$ and any $s \in \mathbb{Z}_{>0}$ such that sB is integral, we have: $\{v \in \mathbb{Z}^{1\times n} \mid vB \in \mathbb{Z}^{1\times m}\} = \{v \in \mathbb{Z}^{1\times n} \mid v(sB) \equiv 0_{1\times m} \bmod s\}$.*

Remark 8. *If $U \in \mathbb{Z}/(s)^{m\times m}$ satisfies $\det U \perp s$, then H is a (minimal) denominator of B if and only if H is a (minimal) denominator of BU. Here, $\perp$ denotes two integers being relatively prime.*

3.2 Smith massagers

Important for our work is the notion of a Smith massager of A.

Definition 9 ([Birmpilis et al. 2023, Definition 1]). *Let $A \in \mathbb{Z}^{n\times n}$ be a nonsingular integer matrix with Smith form S. A matrix $M \in \mathbb{Z}^{n\times n}$ is a* Smith massager *for A if*

(i) it satisfies that

$$AM \equiv 0 \;\mathrm{cmod}\, S, \textit{ and} \tag{6}$$

(ii) there exists a matrix $\hat{W} \in \mathbb{Z}^{n\times n}$ such that

$$\hat{W}M \equiv I_n \;\mathrm{cmod}\, S. \tag{7}$$

It follows directly from Definition 9 that if M is a Smith massager for A, then $\mathrm{cmod}(M, S)$ is also a Smith massager for A. If $M = \mathrm{cmod}(M, S)$, then M is called a *reduced Smith massager*. Compared to A^{-1}, a reduced Smith massager M requires only $O(n^2(\log n + \log ||A||))$ space to store.

The key feature of a Smith massager that we exploit in this paper is the following.

Lemma 10. *Let $A \in \mathbb{Z}^{n\times n}$ be nonsingular with Smith form A. Any Smith massager $M \in \mathbb{Z}^{n\times n}$ for A has the property that MS^{-1} has minimal denominator A.*

The lemma follows directly from Definition 2 combined with [Birmpilis et al. 2023, Theorem 4] which shows that the lattices $\{v \in \mathbb{Z}^{1\times n} \mid vA^{-1} \in \mathbb{Z}^{1\times n}\}$ and $\{v \in \mathbb{Z}^{1\times n} \mid vMS^{-1} \in \mathbb{Z}^{1\times n}\}$ are identical. Instead of working with the rational matrix MS^{-1}, we can avoid fractions using Lemma 7, which shows that, for any s that is a positive multiple of the largest invariant factor of A, the lattices $\{v \in \mathbb{Z}^{1\times n} \mid vMS^{-1} \in \mathbb{Z}^{1\times n}\}$ and $\{v \in \mathbb{Z}^{1\times n} \mid vM(sS^{-1}) \bmod s\}$ are identical. In particular, this implies that $sA^{-1} \equiv_R M(sS^{-1}) \bmod s$.

Example 11. *The input matrix*

$$A = \begin{bmatrix} -8 & 3 & -1 & 0 \\ 0 & 1 & 1 & -1 \\ 4 & -2 & -1 & -1 \\ 4 & -1 & 0 & 0 \end{bmatrix} \in \mathbb{Z}^{4\times 4}$$

has Smith form $S = \mathrm{diag}(1, 1, 1, 16 =: s)$ *and*

$$sA^{-1} = \begin{bmatrix} 2 & 1 & -1 & 9 \\ 8 & 4 & -4 & 20 \\ -8 & 4 & -4 & -12 \\ 0 & -8 & -8 & 8 \end{bmatrix}.$$

A reduced Smith massager for A is given by

$$M = \begin{bmatrix} & & & 1 \\ & & & 4 \\ & & & 4 \\ & & & 8 \end{bmatrix} \in \mathbb{Z}^{4\times 4}.$$

The Hermite form of A is thus the Hermite denominator of the last column of M divided by s. This form is given in Example 3.

Remark 12. *We say that a Smith form diagonal entry is* trivial *if it is equal to 1. It is easy to see that the number of nonzero columns in a reduced Smith massager for A is equal to the number of nontrivial invariant factors of A.*

3.3 Compact representations

In the naive cost model, the integers 0 and 1 both require one bit to store in their binary representation. For example, the total number of bits required to store a nonsingular Hermite form $H \in \mathbb{Z}^{n\times n}$ as a dense $n \times n$ matrix is $O(n^2 + n \log \det H)$ bits, even if $\log \det H \ll n$.

We can save space and simplify the derivation of running time estimates by adopting a data structure that avoids explicitly storing integers that are known *a priori* to be zero, and by avoiding integer multiplications where one of the operands is known *a priori* to be equal to one. For example, we can avoid storing *trivial* column of H (corresponding to diagonal entry $h_i = 1$) or trivial columns of reduced Smith massagers (where $s_i = 1$).

In the proof of the following lemma, recall that we define the bitlength of a vector to be the bitlength of the largest entry in absolute value, as opposed to the sum of the bitlengths of the entries.

Lemma 13. *Let* $H \in \mathbb{Z}^{n\times n}$ *be in Hermite form. Then H can be represented using* $O(n \log \det H)$ *bits by storing the submatrix comprised of its nontrivial columns, together with the list of the indices of the nontrivial columns.*

Proof. Entries in column i of H have magnitude bounded by the diagonal entry h_i of column i. The sum of the bitlengths of the *nontrivial* columns of H is bounded by

$$\sum_{\substack{i=1 \\ h_i \neq 1}}^{n} \lg h_i \quad \leq \quad \sum_{\substack{i=1 \\ h_i \neq 1}}^{n} (1 + \log h_i) \leq \sum_{\substack{i=1 \\ h_i \neq 1}}^{n} (2 \log h_i) = 2 \log \det H.$$

□

A statement similar to Lemma 13 also holds for reduced Smith massagers.

Lemma 14. *Let $M \in \mathbb{Z}^{n\times n}$ satisfy $M = \mathbf{c}\mathrm{mod}(M, S)$ where $S \in \mathbb{Z}^{n\times n}$ is a nonsingular Smith form. Then M can be represented using $O(n \log \det S)$ bits by storing only the nontrivial columns.*

4 COMPUTATIONAL PRELIMINARIES

In this section we define some computational tasks which will be used later in the paper, and derive upper bounds on their complexity. We also summarize in Subsection 4.1 two results which we need from the literature.

We consider first the computation of the remainder modulo Y of the product of two integers. Here, $b \in \mathbb{Z}/(Y)$ implicitly means $b \in [0, Y)$.

Lemma 15. *Let $a \in \mathbb{Z}$ and $b \in \mathbb{Z}/(Y)$. If $\lg a \le D$, then $\mathrm{Rem}(ab, Y)$ can be computed in $O(D(\log Y))$ bit operations.*

Proof. There exists a constant c_1 such that the multiplication ab over $\mathbb{Z}$ has cost bounded by $c_1(\lg a)(\lg b)$. There exists a second constant c_2 such that $\mathrm{Rem}(ab, Y)$ has cost bounded by $c_2(\lg ab/Y)(\lg Y)$. Using $|b| < Y$ shows that both of these cost bounds are bounded by $c(\lg a)(\lg Y)$ where $c = \max(c_1, c_2)$. Using $\lg a \le D$ and $Y > 1$ we have $c(\lg a)(\lg Y) \le cD(1 + \log Y) \le cD(2 \log Y) \in O(D(\log Y))$. □

The following lemma extends Lemma 15 by replacing the first operand a with a matrix, and the second operand b with a vector.

Lemma 16. *Let $A \in \mathbb{Z}^{n\times k}$ and $b \in \mathbb{Z}/(Y)^{k\times 1}$. If the sum of the bitlengths of the columns of A is bounded by D, then $\mathrm{Rem}(Ab, Y)$ can be computed in $O(nD(\log Y))$ bit operations.*

Proof. Decompose A into columns as $A = \begin{bmatrix} \vec{a}_1 & \cdots & \vec{a}_k \end{bmatrix} \in \mathbb{Z}^{n\times k}$, and let d_i be the bitlength of $\vec{a}_i$, $1 \le i \le k$. Then, $\sum_i^k d_i \le D$. Let b_i be entry i of b. Then,

$$\mathrm{Rem}(Ab, Y) = \mathrm{Rem}\left(\sum_{i=1}^{k} \mathrm{Rem}(\vec{a}_i b_i, Y), Y\right).$$

By Lemma 15, there is a constant c such that computing $\mathrm{Rem}(\vec{a}_i b_i, Y) \in \mathbb{Z}/(Y)^{n\times 1}$ has cost bounded by $cnd_i(\log Y)$. Computing all $\mathrm{Rem}(\vec{a}_i b_i, Y)$ then has cost bounded by $\sum_{i=1}^k cnd_i(\log Y) \in O(nD(\log Y))$. Accumulating the sum modulo Y is within this cost. □

The following result follows by accumulating the multiplication cost over the rows of A.

Corollary 17. *Let $A \in \mathbb{Z}^{n\times k}$ and $b \in \mathbb{Z}/(Y)^{k\times 1}$. If the sum of the bitlengths of the rows of A are bounded by D, then $\mathrm{Rem}(Ab, Y)$ can be computed in $O(kD(\log Y))$ bit operations.*

We now apply Lemma 16 to obtain the following result.

Lemma 18. *Given as input*

(i) a nonsingular Smith form $S = \mathrm{diag}(s_1, \ldots, s_n) \in \mathbb{Z}^{n\times n}$,
(ii) a matrix $M \in \mathbb{Z}^{n\times n}$ such that $M = \mathbf{c}\mathrm{mod}(M, S)$, and
(iii) a nonsingular Hermite form $H \in \mathbb{Z}^{n\times n}$,

we can compute $\mathbf{c}\mathrm{mod}(HM, S)$ in $O(n(\log \det S)(\log \det H))$ bit operations.

Proof. Let $\bar{M} = \mathbf{c}\mathrm{mod}(HM, S)$. If $\det S = 1$, then $\bar{M}$ is the zero matrix and there is nothing to compute. Similarly, if $\det H = 1$, then $\bar{M} = M$. Assume therefore that $\det S,\ \det H > 1$. Note that $\bar{M} = \mathbf{c}\mathrm{mod}(M + (H - I)M, S)$. We can thus compute $\bar{M}$ in two steps, by first computing $B := \mathbf{c}\mathrm{mod}((H - I)M, S)$ and then returning $\bar{M} := \mathbf{c}\mathrm{mod}(M + B, S)$.

The second step, which adds together two matrices that are column reduced modulo S, can be done in linear time, that is, in $O(n(\log \det S))$ bit operations. It remains to bound the cost of the first step. By Lemma 13, the sum of the bitlengths of the nonzero columns of $H - I$ are bounded by $2 \log \det H$. Computing B can be done by premultiplying each nontrivial column of M by $(H - I)$, working modulo the corresponding diagonal entry in S. By Lemma 16, there exists a constant c such that the total cost is

$$\sum_{\substack{i=1 \\ s_i \neq 1}}^{n} cn(\log \det H)(\log s_j) \in O(n(\log \det S)(\log \det H))$$

bit operations. □

The following corollary is obtained by replacing the use of Lemma 16 with Corollary 17 in the proof of Lemma 18.

Corollary 19. *Given the same input as in Lemma 18, we can compute* $\mathbf{cmod}(H^T M, S)$ *in* $O(n(\log \det S)(\log \det H))$ *bit operations.*

4.1 Computing Hermite denominators and Smith massagers

We will make use of the following algorithms for computing the Hermite denominator of a rational column vector and fast computation of Smith forms and massagers.

Theorem 20 (Pauderis and Storjohann [2013, Theorem 2]). *There exists an algorithm* `hcol`(w, d) *that takes as input a vector* $w \in \mathbb{Z}/(d)^{n\times 1}$*, and returns as output the Hermite denominator* H *of* wd^{-1}*. The cost of the algorithm is* $O(n(\log d)^2)$ *bit operations. The Hermite form* H *will satisfy* $(\det H) \mid d$*.*

Theorem 21 (Birmpilis et al. [2020, 2023]). *There exists a Las Vegas algorithm* `SmithMassager`(A) *that takes as input a nonsingular* $A \in \mathbb{Z}^{n\times n}$*, and returns as output a tuple* $(M, S, p) \in (\mathbb{Z}^{n\times n}, \mathbb{Z}^{n\times n}, \mathbb{Z}_{>2})$ *with*

(i) S the Smith form of A,
(ii) M is a reduced Smith massager of A, and
(iii) p is prime with $p \perp \det S$ and $\log p \in \Theta(\log n + \operatorname{loglog} ||A||)$.

The algorithm has cost $O(n^3(\log n + \log ||A||)^2(\log n)^2)$ *bit operations, using standard integer and matrix multiplication*

We remark that the prime p in part (iii) of the output specification of Theorem 21 is needed by the subroutine developed in Section 10.

5 DIAGONAL ENTRIES OF THE HERMITE FORM

In this section, we give an algorithm for determining the diagonal entries of the Hermite form of a nonsingular $A \in \mathbb{Z}^{n\times n}$. Let the Smith form of A be S, and suppose M is a Smith massager for A. The algorithm is based on Lemma 10, which states that the Hermite denominator of MS^{-1} is the same as that of A^{-1}.

`HermiteDiagonals`(A, M, S) **Input:** (i) A nonsingular $A \in \mathbb{Z}^{n\times n}$. (ii) The Smith form $S = \mathrm{diag}(s_1, \ldots, s_n)$ of A. (iii) A reduced Smith massager M for A. **Output:** The diagonal entries $h_1, \ldots, h_n$ of the Hermite form of A.

Fig. 2. Problem `HermiteDiagonals`

Theorem 22. *Problem* `HermiteDiagonals` *can be solved in* $O(n(\log \det S)^2)$ *bit operations.*

Proof. Define $s_0 := 1$, and let $0 \le k \le n$ be such that $s_i = 1$ for all $i \le k$. Then, since the first k columns of MS^{-1} are zero, they have minimal denominator I_n, and so can be ignored. By Lemma 6, the following loop will compute matrices $\hat{H}_{k+1}, \hat{H}_{k+2}, \ldots, \hat{H}_n$ in Hermite form such that $\hat{H}_n\hat{H}_{n-1}\cdots\hat{H}_{k+1}$ is a minimal denominator of MS^{-1}.

for $i = k+1$ **to** n **do**
  $\hat{H}_i :=$ `hcol`$(\mathrm{Column}(M, i), s_i)$
  $M := \mathbf{cmod}(\hat{H}_i M, S)$
od

By Theorem 20, the cost of the call to `hcol` in iteration i is bounded by $cn(\log s_i)^2$ for some constant c. The total cost of all calls to `hcol` is therefore $O(n(\log S)^2)$. By Lemma 16, the cost of updating M during iteration i is bounded by $\hat{c}n(\log \det \hat{H}_i)(\log \det S)$ bit operations for some constant $\hat{c}$. Since $\det \hat{H}_i \mid s_i$, this is bounded by $\hat{c}n(\log s_i)(\log \det S)$. The total cost of all updates of M is then also $O(n(\log \det S)^2)$.

While the product $\hat{H}_n\hat{H}_{n-1}\cdots\hat{H}_{k+1}$ is a minimal denominator of MS^{-1} that is upper triangular, it might not be in Hermite form because the off-diagonal entries might not be reduced. However the diagonal entries of $\hat{H}_n\hat{H}_{n-1}\cdots\hat{H}_{k+1}$ will be the same as those of H. Taking advantage of our compact representation for the $\hat{H}_*$, the total cost of computing the diagonal entries of H is then bounded by $O(n(\log \det S)^2)$. □

6 COLUMN HOWELL FORMS

Working with matrix denominators, as discussed in Subsection 3.1, naturally implies doing linear algebra in the residue class ring $\mathsf{R} := \mathbb{Z}/(s)$ for a given modulus $s \in \mathbb{Z}_{>0}$ (c.f. Lemma 7). In this section, we investigate a type of column echelon form for matrices in such a residue ring.

For a matrix $B \in \mathsf{R}^{n\times n}$, we denote by

$$\mathrm{Span}(B) = \{Bv \in \mathsf{R}^{n\times 1} \mid v \in \mathsf{R}^{n\times 1}\}$$

the set of all R-linear combinations of the columns of B. By $\mathrm{Span}_k(B)$ we denote the subset of $\mathrm{Span}(B)$ consisting of all column vectors that have the last k entries zero.

A column Howell form of B, first introduced by Howell [1986], is a matrix $T \in \mathsf{R}^{n\times n}$ that is right equivalent to B over R and that satisfies the *Howell property*: for all $0 \le k \le n$, $\mathrm{Span}_k(B) = \mathrm{Span}(T_k)$ where T_k is the submatrix of T comprised of those columns that have the last k entries zero.

Example 23. *Consider the matrix*

$$B = \begin{bmatrix} & & & 1 \\ & & & 4 \\ & & & 4 \\ & & & 8 \end{bmatrix} \in \mathbb{Z}/(16)^{4\times 4}.$$

The span of the columns of B which have the last entry zero (in this example the first three zero columns) contains only the zero vector. But multiplying the last column of B by 2 yields the nonzero column

$$\begin{bmatrix} 2 \\ 8 \\ 8 \\ \end{bmatrix},$$

with last entry zero, and so B does not satisfy the Howell property. In this case the column triangularization of B given by

$$T = \begin{bmatrix} & 4 & 2 & 1 \\ & & 8 & 4 \\ & & 8 & 4 \\ & & & 8 \end{bmatrix} = \begin{bmatrix} & & & 1 \\ & & & 4 \\ & & & 4 \\ & & & 8 \end{bmatrix} \begin{bmatrix} 1 & & & 1 \\ & 1 & 13 & \\ & & 1 & 1 \\ 4 & 0 & 6 & 11 \end{bmatrix} = BU \tag{8}$$

with U unimodular does satisfy the Howell property.

The Howell form is a natural generalization of the notion of the column echelon form over a field. Variations include alternate locations for the zero columns and/or including some additional normalization conditions. For our purposes, in order to simplify the subsequent presentation, we say that a matrix T is in *Howell form* if T satisfies the Howell property and is upper triangular with the diagonal entries being positive and divisors of the modulus s. The diagonal entries of the zero columns modulo s are replaced with s in order to be positive. Uniqueness of the form can be achieved by stipulating that off-diagonal entries are reduced modulo the diagonal entry in the same row, as per Howell [1986, Theorem 2], but we do not require this. We will however use the fact that the diagonal entries of a Howell form are unique.

Example 24. *Consider the matrix $B \in \mathbb{Z}/(16)^{4\times n}$ from Example 23. A Howell form of B is obtained from the matrix T in (8) by swapping the first two columns and adding the pivot 16 in the second column:*

$$\begin{bmatrix} 4 & 0 & 2 & 1 \\ & 16 & 8 & 4 \\ & & 8 & 4 \\ & & & 8 \end{bmatrix}.$$

Another Howell form of B is

$$\begin{bmatrix} 4 & 0 & 6 & 11 \\ & 16 & 8 & 12 \\ & & 8 & 12 \\ & & & 8 \end{bmatrix}.$$

7 SOLVING IN THE DUAL: HERMITE VIA HOWELL

Throughout this section, let

$$H = \begin{bmatrix} h_1 & h_{12} & \cdots & h_{1n} \\ & h_2 & \cdots & h_{2n} \\ & & \ddots & \vdots \\ & & & h_n \end{bmatrix} \in \mathbb{Z}^{n\times n}$$

be the Hermite form of $A \in \mathbb{Z}^{n\times n}$. For $1 \le j \le n$, define

$$H_j = \begin{bmatrix} 1 & & & h_{1,j} & & & \\ & \ddots & & \vdots & & & \\ & & 1 & h_{j-1,j} & & & \\ & & & h_j & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix} \in \mathbb{Z}^{n\times n}$$

to be the $n \times n$ matrix with column j equal to that of H and the remaining columns those of I_n. Computing H is thus equivalent to computing $H_1, \ldots, H_n$. In addition, it is useful to note that the first j columns of $H_j \cdots H_1$ are those of H, while the last $n - j$ columns are those of I_n.

In this section, we establish a duality between H and any Howell form T of sA^{-1} over $\mathbb{Z}/(s)$, with s a positive integer such that sA^{-1} is integral. In particular, we show that column j of $(H_{j-1} \cdots H_1)T$ is congruent modulo s to

$$-\frac{s}{h_j}\begin{bmatrix} h_{1j} \\ \vdots \\ h_{j-1,j} \\ -1 \\ \\ \\ \end{bmatrix} \bmod s.$$

This property points out the following algorithm for computing H:

for $j = 1$ **to** n **do**
  Recover H_j from column j of T
  $T := \mathrm{Rem}(H_j T, s)$
od

We first show that any nonsingular upper triangular matrix over $\mathbb{Z}$ corresponds to a Howell form over $\mathbb{Z}/(s)$.

Lemma 25. *Let $T \in \mathbb{Z}^{n\times n}$ be nonsingular and upper triangular. If $s \in \mathbb{Z}_{>0}$ is such that sT^{-1} is integral, then sT^{-1} satisfies the Howell property over $\mathbb{Z}/(s)$.*

Proof. To establish the Howell property, we need to show that, for $0 \le k \le n$, $\mathrm{Span}_k(sT^{-1})$ is equal to the span of the columns of sT^{-1} that have the last $n - k$ entries zero. To this end, fix k and decompose T as

$$T = \begin{bmatrix} T_1 & \bar{T} \\ & T_2 \end{bmatrix}$$

where $T_2 \in \mathbb{Z}^{k\times k}$ and the dimensions of T_1 and $\bar{T}$ are implied. Then,

$$sT^{-1} = \begin{bmatrix} sT_1^{-1} & -sT_1^{-1}\bar{T}T_2^{-1} \\ & sT_2^{-1} \end{bmatrix} \in \mathbb{Z}^{n\times n},$$

and it will suffice to show that

$$\mathrm{Span}_k(sT^{-1}) \subseteq \mathrm{Span}\left(\begin{bmatrix} sT_1^{-1} \\ \end{bmatrix}\right).$$

This is equivalent to saying that for any vector $v \in \mathbb{Z}^{n\times 1}$ such that

$$sT^{-1}v = \begin{bmatrix} \bar{v} \\ \end{bmatrix} \bmod s,$$

for some $\bar{v} \in \mathbb{Z}^{(n-k)\times 1}$, there exists another vector $u \in \mathbb{Z}^{(n-k)\times 1}$ such that $sT_1^{-1}u = \bar{v}$. Now,

$$\begin{aligned} sT^{-1}v &= \begin{bmatrix} sT_1^{-1} & -sT_1^{-1}\bar{T}T_2^{-1} \\ & sT_2^{-1} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \\ &= \begin{bmatrix} sT_1^{-1}v_1 - sT_1^{-1}\bar{T}T_2^{-1}v_2 \\ sT_2^{-1}v_2 \end{bmatrix} && (9) \\ &= \begin{bmatrix} \bar{v} \\ \end{bmatrix} \bmod s. && (10) \end{aligned}$$

From the lower block of (9) and (10), it follows that there exist a vector $v_2' \in \mathbb{Z}^{k\times 1}$ such that

$$sT_2^{-1}v_2 = sv_2' \iff v_2 = T_2 v_2'.$$

Moreover, from the upper block of (9) and (10), we have that

$$\begin{aligned} \bar{v} &= sT_1^{-1}v_1 - sT_1^{-1}\bar{T}T_2^{-1}v_2 \\ &= sT_1^{-1}v_1 - sT_1^{-1}\bar{T}v_2' \\ &= sT_1^{-1}(v_1 - \bar{T}v_2'), \end{aligned}$$

which proves the claim. □

Corollary 26. *If H is the Hermite form of A, then sH^{-1} is a Howell form of sA^{-1} over $\mathbb{Z}/(s)$.*

Proof. The result follows since $sH^{-1} \equiv_R sA^{-1}$, sH^{-1} is upper triangular, the diagonal entries of sH^{-1} are positive divisors $s/h_1, s/h_2, \ldots, s/h_n$ of s and, from Lemma 25, sH^{-1} satisfies the Howell property. □

Corollary 27. *The diagonal entries of any any Howell form of sA^{-1} over $\mathbb{Z}/(s)$ are equal to $s/h_1, \ldots, s/h_n$.*

Proof. This follows from Corollary 26 and the fact that the diagonal entries of a Howell form of sA^{-1} are unique. □

Lemma 28. *Let T be a Howell form of sA^{-1} over $\mathbb{Z}/(s)$. Then, $H_j \cdots H_1$ is a denominator of the first j columns of $(1/s)T$, for $1 \le j \le n$.*

Proof. Since T is right equivalent to sA^{-1} over $\mathbb{Z}/(s)$, and H is a denominator of A^{-1}, we have that H is a denominator of $(1/s)T$. The claim in the lemma now follows from the fact that T is upper triangular. In particular, premultiplying an upper triangular matrix by H_k for $k > j$ does not change the first j columns. Let $T_{1...j}$ denote the submatrix of T comprised of the first j columns. Then ,

$$\begin{aligned} HT_{1..j} &= (H_n \cdots H_{j+1})(H_j \cdots H_1)T_{1..j} && (11) \\ &= (H_j \cdots H_1)T_{1..j}. && (12) \end{aligned}$$

Since the left hand side of (11) is zero modulo s, so is the right hand side of (12). □

Theorem 29. *Let T be a Howell form of sA^{-1} over $\mathbb{Z}/(s)$. Then, $H_j \cdots H_1$ is the minimal Hermite denominator of the first j columns of $(1/s)T$, for $1 \le j \le n$.*

Proof. Recall that we let $T_{1...j}$ denote the first j columns of T. We will use induction. For $j = 1$, the claim of the theorem follows from Corollary 27 and Lemma 28, since the first diagonal entry of T is s/h_1 and H_1 is a denominator of $(1/s)T_1$.

Now, assume that the claim is true for $j - 1$, for some $j > 1$, that is, assume that $H_{j-1} \cdots H_1$ is the minimal Hermite denominator of $(1/s)T_{1...j-1}$. Then, let v be column j of $(H_{j-1} \cdots H_1)T$. We first show that v has the shape

$$v = \begin{bmatrix} * \\ \vdots \\ * \\ s/h_j \\ \\ \\ \end{bmatrix} \in \mathbb{Z}/(s)^{n\times 1}. \tag{13}$$

To see this, note that premultiplying T by $H_{j-1} \cdots H_1$ only affects the first $j - 1$ rows, so the last $n - j + 1$ entries of v are the same as those of column j of T. By Lemma 27, entry j of v is equal to s/h_j.

Next, by Lemma 6, a minimal denominator of $(1/s)T_{1...j}$ is given by $\bar{H}_j H_{j-1} \cdots H_1$, where $\bar{H}_j$ is the minimal Hermite denominator of $(1/s)v$. By Lemma 28, $H_j H_{j-1} \cdots H_1$ is a denominator of $(1/s)T_{1...j}$, so we must have that $\det \bar{H}_j$ is a divisor of $\det H_j = h_j$. But since entry j of $(1/s)v$ is $1/h_j$, the diagonal entry j of $\bar{H}_j$ must equal h_j, which means that the remaining columns of $\bar{H}_j$ have diagonal entry 1. Because of the shape of $\bar{H}_j$, and the fact that it is in Hermite form, we have that $\bar{H}_j H_{j-1} \cdots H_1$ is also in Hermite form. The uniqueness of the Hermite form then implies that $\bar{H}_j = H_j$. □

Corollary 30. *For $1 \le j \le n$, column j of $(H_{j-1} \cdots H_1)T$ is equal to*

$$-\frac{s}{h_j} \begin{bmatrix} h_{1j} \\ \vdots \\ h_{j-1,j} \\ -1 \\ \\ \\ \end{bmatrix} \bmod s. \tag{14}$$

Proof. Column j of $(H_{j-1} \cdots H_1)T$ is the vector $v \in \mathbb{Z}/(s)^{n\times 1}$ in (13), from the proof of Theorem 29, where it was established that entry j of v is s/h_j, the last $n - j$ entries of v are zero, and H_j is the minimal Hermite denominator of $(1/s)v$. The only such vector v is the one shown in (14), namely, column j of sH_j^{-1}. □

The following example illustrates the approach of Corollary 30 for computing the Hermite form over $\mathbb{Z}$ by first computing a Howell form in the space $\mathbb{Z}/(s)$.

Example 31. *The input matrix*

$$A = \begin{bmatrix} -8 & 3 & -1 & 0 \\ 0 & 1 & 1 & -1 \\ 4 & -2 & -1 & -1 \\ 4 & -1 & 0 & 0 \end{bmatrix} \in \mathbb{Z}^{4\times 4}$$

has Smith form $S = \mathrm{diag}(1, 1, 1, 16 =: s)$ *and*

$$sA^{-1} = \begin{bmatrix} 2 & 1 & -1 & 9 \\ 8 & 4 & -4 & 20 \\ -8 & 4 & -4 & -12 \\ 0 & -8 & -8 & 8 \end{bmatrix}.$$

We now work over $\mathbb{Z}/(s)$. *A Howell form of* sA^{-1} *over* $\mathbb{Z}/(s)$ *is given by*

$$T = \begin{bmatrix} 4 & 0 & 6 & 11 \\ & 16 & 8 & 12 \\ & & 8 & 12 \\ & & & 8 \end{bmatrix} = \begin{bmatrix} \frac{s}{4} & 0 & 6 & 11 \\ & \frac{s}{1} & 8 & 12 \\ & & \frac{s}{2} & 12 \\ & & & \frac{s}{2} \end{bmatrix} = \begin{bmatrix} \frac{s}{h_1} & 0 & 6 & 11 \\ & \frac{s}{h_2} & 8 & 12 \\ & & \frac{s}{h_3} & 12 \\ & & & \frac{s}{h_4} \end{bmatrix}.$$

The diagonal elements of H *are thus* $h_1, h_2, h_3, h_4 = 4, 1, 2, 2$. *Using Corollary 30 gives the following:*

$$j = 1: \qquad H_1 = \begin{bmatrix} 4 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \text{ and } H_1T = \begin{bmatrix} 0 & 8 & 12 \\ 16 & 8 & 12 \\ & 8 & 12 \\ & & 8 \end{bmatrix}$$

$$j = 2: \qquad H_2 = \begin{bmatrix} 1 & 0 & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \text{ and } H_2H_1T = H_1T$$

$$j = 3: \qquad H_3 = \begin{bmatrix} 1 & & 1 & \\ & 1 & 1 & \\ & & 2 & \\ & & & 1 \end{bmatrix} \text{ and } H_3H_2H_1T = \begin{bmatrix} & & & 8 \\ & & & 8 \\ & & & 8 \\ & & & 8 \end{bmatrix}$$

$$j = 4: \qquad H_4 = \begin{bmatrix} 1 & & & 1 \\ & 1 & & 1 \\ & & 1 & 1 \\ & & & 2 \end{bmatrix} \text{ and } H_4H_3H_2H_1T = 0_{4\times 4}$$

The Hermite denominator of $(1/s)T$ *is thus*

$$H_4H_3H_2H_1 = \begin{bmatrix} 4 & 0 & 1 & 1 \\ & 1 & 1 & 1 \\ & & 2 & 1 \\ & & & 2 \end{bmatrix}.$$

8 COMPUTING A HERMITE FORM FROM A HOWELL FORM

For $A \in \mathbb{Z}^{n\times n}$ nonsingular with Smith form S, let $s = s_n$ be the largest invariant factor of S, and let $S^* = sS^{-1}$ and $A^* = sA^{-1}$. A problem with using the approach of Example 31 to compute H, is that the size of A^* and its Howell form T over $\mathbb{Z}/(s)$ can be $\Omega(n^2 \log s)$ bits.

In this section, we show how we can avoid computing the Howell form T explicitly, and instead work with matrices $M, U \in \mathbb{Z}^{n\times n}$ such that

$$T = MS^*U \bmod s.$$

We start first with MS^*, where M is a reduced Smith massager for A, which we know is right equivalent to A^* over $\mathbb{Z}/(s)$ but has total size only $O(n \log \det S)$ bits, as per Lemma 14. We then compute a transformation matrix U such that $T = MS^*U \bmod s$.

Lemma 32. *Let $U \in \mathbb{Z}/(s)^{n\times n}$ be such that $T = MS^*U$ is a Howell form of MS^* over $\mathbb{Z}/(s)$. Then $T = MS^*\,\mathbf{r}\mathrm{mod}(U, S)$.*

Thus, we may assume without loss of generality that $U = \mathbf{r}\mathrm{mod}(U, S)$. So, while the overall size of T itself can be large, the transformation matrix U to generate T can be assumed to be small, that is, just like M, it can be represented using $O(n \log \det S)$ bits. The following example illustrates how a Howell form T can be represented implicitly as the product MS^*U.

Example 33. *The input matrix*

$$A = \begin{bmatrix} 2 & -1 & & & \\ & 2 & -1 & & \\ & & 2 & \ddots & \\ & & & \ddots & -1 \\ & & & & 2 \end{bmatrix} \in \mathbb{Z}^{n\times n}$$

has Smith form $\mathrm{diag}(1, \ldots, 1, 2^n =: s)$ *and*

$$A^* := sA^{-1} = \begin{bmatrix} 2^{n-1} & 2^{n-2} & 2^{n-3} & \cdots & 1 \\ & 2^{n-1} & 2^{n-2} & \cdots & 2^1 \\ & & 2^{n-1} & \cdots & 2^2 \\ & & & \ddots & \vdots \\ & & & & 2^{n-1} \end{bmatrix}.$$

By Lemma 25, A^* is in Howell form over $\mathbb{Z}/(s)$. The sum of the bitlengths of entries in A^* is clearly $\Theta(n^3)$.

However, a reduced Smith massager for A is given by the $n \times n$ matrix

$$M = \begin{bmatrix} & & & & 1 \\ & & & & 2^1 \\ & & & & 2^2 \\ & & & & \vdots \\ & & & & 2^{n-1} \end{bmatrix}.$$

Let $S^* := sS^{-1}$. A matrix U such that $A^* = MS^*U$ is given by

$$U = \begin{bmatrix} & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ 2^{n-1} & 2^{n-2} & 2^{n-3} & \cdots & 2^1 & 1 \end{bmatrix}.$$

The sum of the bitlengths of all entries in M and U is only $O(n^2)$. Restricting M and U to their nonzero columns and rows, respectively, and restricting S^* to its only nonzero entry, gives

$$A^* = MS^*U = \begin{bmatrix} 1 \\ 2^1 \\ 2^2 \\ \vdots \\ 2^{n-1} \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} 2^{n-1} & 2^{n-2} & 2^{n-3} & \cdots & 2^1 & 1 \end{bmatrix} \mod s.$$

Instead of working with an explicit Howell form T of A^*, we work with the right hand side of the equation $T = MS^*U$. At iteration j, we then compute column j of $-(h_j/s)\mathrm{Rem}(MS^*U, s) \in \mathbb{Z}/(h_j)^{n\times 1}$ which gives the off-diagonal entries in column j of H. Finally, to update T at iteration j we simply update $M := \mathbf{c}\mathrm{mod}(H_jM, S)$.

Example 34. *The input matrix*

$$A = \begin{bmatrix} -8 & 3 & -1 & 0 \\ 0 & 1 & 1 & -1 \\ 4 & -2 & -1 & -1 \\ 4 & -1 & 0 & 0 \end{bmatrix} \in \mathbb{Z}^{4\times 4}$$

has Smith form $S = \mathrm{diag}(1, 1, 1, 16 =: s)$. Since in this example A has only one nontrivial invariant factor, a reduced Smith massager M for A and transformation matrix U such that MS^*U is in Howell form will have one nonzero column and row respectively. Restrict M to its last column, U to its last row, set $S = \mathrm{diag}(16)$ and $S^* = \mathrm{diag}(1)$. Then

$$T = MS^*U = \begin{bmatrix} 1 \\ 4 \\ 4 \\ 8 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} 4 & 0 & 6 & 11 \end{bmatrix}.$$

Suppose we have precomputed the diagonal entries $h_1, h_2, h_3, h_4 = 4, 1, 2, 2$ of the Hermite denominator of MS^{-1}. Applying the approach of Corollary 30 gives the following:

$$j = 1: \quad -\tfrac{h_1}{s}\mathrm{Column}(MS^*U, 1) = \begin{bmatrix} -1 \\ \\ \\ \\ \end{bmatrix} \quad M := \mathbf{c}\mathrm{mod}(H_1M, S) = \begin{bmatrix} 4 \\ 4 \\ 4 \\ 8 \end{bmatrix}$$

$$j = 2: \quad -\tfrac{h_2}{s}\mathrm{Column}(MS^*U, 2) = \begin{bmatrix} 0 \\ -1 \\ \\ \\ \end{bmatrix} \quad M := \mathbf{c}\mathrm{mod}(H_2M, S) = \begin{bmatrix} 4 \\ 4 \\ 4 \\ 8 \end{bmatrix}$$

$$j = 3: \quad -\tfrac{h_3}{s}\mathrm{Column}(MS^*U, 3) = \begin{bmatrix} 1 \\ 1 \\ -1 \\ \\ \end{bmatrix} \quad M := \mathbf{c}\mathrm{mod}(H_2M, S) = \begin{bmatrix} 8 \\ 8 \\ 8 \\ 8 \end{bmatrix}$$

$$j = 4: \quad -\tfrac{h_4}{s}\mathrm{Column}(MS^*U, 4) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \quad M := \mathbf{c}\mathrm{mod}(H_3M, S) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

The Hermite basis of A is thus given by

$$H_4H_3H_2H_1 = \begin{bmatrix} 4 & 0 & 1 & 1 \\ & 1 & 1 & 1 \\ & & 2 & 1 \\ & & & 2 \end{bmatrix}.$$

`HermiteViaHowell`$(A, M, S, U, [h_1, \dots, h_n], p)$
Input:
(i) A nonsingular $A \in \mathbb{Z}^{n\times n}$.
(ii) The Smith form $S = \mathrm{diag}(s_1, \dots, s_n)$ of A.
Let $s := s_n$ and $S^* := sS^{-1}$.
(iii) A reduced Smith massager M for A.
(iv) A $U \in \mathbb{Z}^{n\times n}$ such that $\mathrm{Rem}(MS^*U, s)$ is in Howell form over $\mathbb{Z}/(s)$ and $U = \mathbf{r}\mathrm{mod}(U, S)$.
(v) The diagonal entries $h_1, \dots, h_n$ of the Hermite form of A.
(vi) A prime p that satisfies $p \perp s$ and $\log p \in O(\mathrm{loglog}\, S)$.
Output:
The Hermite form H of A.

Fig. 3. Problem `HermiteViaHowell`

Theorem 35. *Problem* `HermiteViaHowell` *can be solved in*

$$O(n(\log\det S)^2 + n^2(\log\det S)(\mathrm{loglog}\det S))$$

bit operations.

Proof. By Corollary 30, we can compute $H_1, \dots, H_n$ iteratively as follows:

for $j = 1$ **to** n **do**
  # If $h_j = 1$ then set $H_j := I_n$ and go to next loop iteration.
  1. # Let $u \in \mathbb{Z}/(s)^{1\times n}$ be column j of U.
  $v := -(h_j/s)\mathrm{Rem}(MS^*u, s)$
  2. # Construct H_j from v and h_j.
  $M := \mathbf{c}\mathrm{mod}(H_jM, s)$
od

For the construction of H_j in Step 2, the off-diagonal entries in column j are given by the first $j-1$ entries of v, and h_j is given as input. The proof of Theorem 22 shows that the total cost of the updates to M in Step 2 is bounded by $O(n(\log\det S)^2)$ bit operations.

In Section 10 we develop an algorithm `ScaledMatVecProd` that will compute v in Step 1 during iteration j with the call

$$v := \texttt{ScaledMatVecProd}(M, S, u, h_j, p).$$

The `ScaledMatVecProd` algorithm exploits the properties $M = \mathbf{c}\mathrm{mod}(M, S)$, $u = \mathbf{r}\mathrm{mod}(u, s)$, the product MS^*u is only required modulo s, and that $\mathrm{Rem}(MS^*u, s)$ has a known factor s/h_j. We show later in Theorem 39, that `ScaledMatVecProd` has cost

$$O(n(\log\det S)(\log h_j + \mathrm{loglog}\det S) + (\log\det S)^2) \tag{15}$$

bit operations. Since $\prod_{i=1}^{n} h_j = \det S$, the sum of (15) over over all h_j with $j > 1$ is bounded by the cost stated in the theorem. □

9 COMPUTING THE MULTIPLIER FOR A HOWELL FORM

In this section, we work over the residue class ring $\mathrm{R} = \mathbb{Z}/(s)$ for a given modulus $s \in \mathbb{Z}_{>0}$. Howell [1986] gives an algorithm to compute a Howell form T of a $B \in \mathrm{R}^{n\times n}$. Here we adapt Howell's approach to our context. In particular, instead of T, we focus on the invertible transformation matrix $U \in \mathrm{R}^{n\times n}$ such that $T = BU$. Furthermore, we know positive divisors $h_1, \ldots, h_n$ of s such that the diagonal entries of the Howell form are $t_1, \ldots, t_n$, with $t_i = s/h_i$.

Howell's algorithm begins by augmenting the input matrix with n initial zero columns: to this end, let $\bar{B} := \left[\begin{array}{c|c} 0_{n\times n} & B \end{array}\right] \in \mathrm{R}^{n\times 2n}$. Our goal now is to find a matrix $\bar{U} \in \mathrm{R}^{2n\times 2n}$ such that $\bar{B}\bar{U}$ is a Howell form of $\bar{B}$, as defined before Example 24. Once $\bar{U}$ has been found, we can take U to be the trailing principal $n \times n$ submatrix of $\bar{U}$. Then BU will be a Howell form of B.

Howell's algorithm proceeds in n iterations, for $i = 0, 1, \ldots, n-1$. We initialize $\bar{U} = I_{2n}$. At the start of iteration $i = 0$ we thus have $\bar{B}\bar{U} = \bar{B}$. By the time we reach the start of iteration i, the matrix $\bar{U}$ has been updated so that

$$\bar{B}\bar{U} = \left[\begin{array}{c|cccc|ccc} & * & \cdots & * & * & * & \cdots & * \\ & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ & t_{n-i}a_1 & \cdots & t_{n-i}a_{n-1} & t_{n-i}a_n & * & \cdots & * \\ \hline & & & & & t_{n-i+1} & \cdots & * \\ & & & & & & \ddots & \vdots \\ & & & & & & & t_n \end{array}\right]. \tag{16}$$

Note that we do not compute the complete partial triangularization $\bar{B}\bar{U}$ in (16). We will see that we only need the elements $a_1, \ldots, a_n \in \mathbb{Z}/(h_{n-i})$ shown in (16). Since we are working modulo s and $h_{n-i} = s/t_{n-i}$, the integers $a_1, \ldots, a_n$ can be considered to be elements of $\mathbb{Z}/(h_{n-i})$. Iteration i now applies the following two-part unimodular column transformation. Howell [1986] points out that there exist integers $c_1, \ldots, c_{n-1}, c_n \in \mathbb{Z}/(h_{n-i})$, with c_n relatively prime to s, satisfying

$$c_1a_1 + \cdots + c_{n-1}a_{n-1} + c_na_n = 1 \bmod h_{n-i}.$$

Postmultipying the matrix on the right of (16) by the matrix

$$C_i = \left[\begin{array}{c|cccc|c} I_{n-i} & & & & & \\ \hline & 1 & & & c_1 & \\ & & \ddots & & \vdots & \\ & & & 1 & c_{n-1} & \\ & & & & c_n & \\ \hline & & & & & I_i \end{array}\right] \tag{17}$$

gives

$$\left[\begin{array}{c|cccc|ccc} & * & \cdots & * & * & * & \cdots & * \\ & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ & t_{n-i}a_1 & \cdots & t_{n-i}a_{n-1} & t_{n-i} & * & \cdots & * \\ \hline & & & & & t_{n-i+1} & \cdots & * \\ & & & & & & \ddots & \vdots \\ & & & & & & & t_n \end{array}\right]. \tag{18}$$

Thus, we can use t_{n-i} to zero out the nonzero entries to the left of t_{n-i} This step also fills in a new column which is to be used in the subsequent iterations. If we were to postmultiply the matrix in (18) by

$$W_i := \left[\begin{array}{cc|ccccc|c} I_{n-i-1} & & & & & & & \\ & 1 & & & & & & \\ \hline & & 1 & & & & & \\ & & & \ddots & & & & \\ & & & & & 1 & & \\ & h_{n-i} & -a_1 & \cdots & & -a_{n-1} & 1 & \\ \hline & & & & & & & I_i \end{array}\right] \tag{19}$$

then we would obtain

$$\left[\begin{array}{c|cccc|ccc} * & * & \cdots & * & * & * & \cdots & * \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ * & * & \cdots & * & * & * & \cdots & * \\ & & & & t_{n-i} & * & \cdots & * \\ \hline & & & & & t_{n-i+1} & \cdots & * \\ & & & & & & \ddots & \vdots \\ & & & & & & & t_n \end{array}\right].$$

Note that we can do these triangularizations implicitly as we only really need the a_*. The final computational part, at step i, is to update $\bar{U} := \bar{U}C_iW_i$.

At iteration i, the Howell transform algorithm thus has three steps:

(1) Compute the entries $\left[\begin{array}{ccc} a_1 & \cdots & a_n \end{array}\right] \in \mathbb{Z}/(h_{n-i})^{1\times n}$ of (16).
(2) Compute the matrices C_i and W_i.
(3) Update $\bar{U} := \bar{U}C_iW_i$.

Note that if $h_{n-i} = 1$ then iteration i can be skipped since C_i and W_i will be the identity matrices. For Step (2) we can appeal to the following result.

Lemma 36. *Given integers $h_{n-i} \in \mathbb{Z}_{>0}$ and $a_1, \ldots, a_n \in \mathbb{Z}/(h_{n-i})$, we can compute the off-diagonal nonzero entries of matrices $C_i, W_i \in \mathbb{Z}^{2n\times 2n}$ as seen in (17) and (19), respectively, in time $O(n(\log h_{n-i})^2)$.*

Proof. Storjohann and Mulders [1998, Lemma 2] show that the $c_1, \ldots, c_{n-1}$ can be computed in the allotted time, with c_n just an extra gcd operation over $\mathbb{Z}/(h_{n-i})$. Computing the entries of W_i just involves negating the a_i. □

For the analysis of Steps (1) and (3), we consider the special case of an input matrix $B = MS^*$ as specified in Figure 4.

`SpecialHowellTransform`$(A, M, S, [h_1, \ldots, h_n], p)$
Input:
(i) A nonsingular $A \in \mathbb{Z}^{n\times n}$.
(i) The Smith form $S = \operatorname{diag}(s_1, \ldots, s_n)$ of A.
Let $s := s_n$ and $S^* := sS^{-1}$.
(ii) A reduced Smith massager M for A.
(iii) The diagonal entries $h_1, \ldots, h_n$ of the Hermite form of A.
(iv) A prime p that satisfies $p \perp s$ and $\log p \in \Theta(\operatorname{loglog} s)$.
Output:
A matrix $U = \mathbf{r}\mathrm{mod}(U, S) \in \mathbb{Z}^{n\times n}$ such that MS^*U is a Howell form of MS^* over $\mathbb{Z}/(s)$.

Fig. 4. Problem `SpecialHowellTransform`

THEOREM 37. *Problem* `SpecialHowellTransform` *can be solved in in*

$$O(n(\log\det S)^2 + n^2(\log\det S)(\operatorname{loglog}\det S))$$

bit operations.

PROOF. We adapt Howell's algorithm described at the start of this section to compute an $n \times 2n$ matrix $\bar{U}$ such $MS^*\bar{U} = \begin{bmatrix} 0_{n\times n} & T \end{bmatrix}$, with T a Howell form of MS^* over $\mathbb{Z}/(s)$. Our output U is thus the submatrix comprised of the last n columns of $\bar{U}$. Because of the presence of the scaling matrix S^*, we can keep the rows of $\bar{U}$ reduced modulo the corresponding diagonal entries in S. In other words, we maintain $\bar{U} = \mathbf{r}\mathrm{mod}(\bar{U}, S)$ throughout the algorithm.

Initialize $\bar{U} = \left[\begin{array}{c|c} 0_{n\times n} & I_n \end{array}\right]$. We perform n iterations for $i = 0, 1, \ldots, n-1$. At the start of iteration i the matrix $MS^*\bar{U}$ has exactly the shape shown in (16). Like before, iteration i consists of three steps:

(1) Compute the entries $\begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} \in \mathbb{Z}/(h_{n-i})^{1\times n}$ of (16).
(2) Compute the matrices C_i and W_i.
(3) Update $\bar{U} := \mathbf{r}\mathrm{mod}(\bar{U}C_i, S)$ and then $\bar{U} := \mathbf{r}\mathrm{mod}(\bar{U}W_i, S)$.

At iteration i, the computation of Step 1 aligns with the specification of the `ScaledMatVecProd` subroutine that is later developed in Section 10. In particular, the output of

$$\texttt{ScaledMatVecProd}(M', S, u', h_{n-i}, p),$$

where

- M' is the transpose of the submatrix of $\bar{U}$ containing columns from $(n-i+1)$ to $(2n-i)$, and
- u' is the transpose of row $n-i$ of M,

contains exactly the a_i's we want. The cost of this call to `ScaledMatVecProd` is

$$O(n(\log\det S)(\log h_{n-j} + \operatorname{loglog}\det S) + (\log\det S)^2) \tag{20}$$

bit operations (Theorem 39).

By Lemma 36, the cost of Step 2 is

$$O(n(\log h_{n-i})^2) \tag{21}$$

bit operations.

Finally, the two multiplications in Step 3, namely, $\mathbf{r}\mathrm{mod}(\bar{U}C_i, S)$ and $\mathbf{r}\mathrm{mod}((\bar{U}C_i)W_i, S)$, are covered by Corollary 19 and Lemma 18, respectively, and have cost bounded by

$$O(n(\log\det S)(\log h_{n-i})) \tag{22}$$

bit operations.

Summing (20), (21) and (22) over all iterations i with $h_{n-i} > 1$ gives the cost bound stated in the theorem. ◻

Example 38. *Let*

$$A = \begin{bmatrix} -13 & 10 & -20 & 27 \\ 27 & 30 & 15 & 30 \\ 0 & 15 & 15 & 6 \\ -21 & 0 & -15 & 9 \end{bmatrix},$$

with Smith form $S = \operatorname{diag}(1, 3, 15, 105)$ *and reduced Smith massager*

$$M = \begin{bmatrix} 0 & 2 & 0 & 55 \\ 0 & 0 & 7 & 32 \\ 0 & 2 & 2 & 41 \\ 0 & 2 & 10 & 10 \end{bmatrix}$$

be given. We are also given the diagonal entries $h_1, h_2, h_3, h_4 = 1, 15, 15, 21$ *of the Hermite form of* A. *Let* $S^* = sS^{-1}$ *with* $s = 105$. *We illustrate the method used in the proof of Theorem 37 to compute a matrix* U *such that* $T = MS^*U$ *is in Howell form over* $\mathbb{Z}/(s)$. *Note that we know that* T *will have diagonal entries* $t_1, t_2, t_3, t_4 = 105, 7, 7, 5$*, that is,* $t_i = s/h_i$.

Initialize $\bar{U} = \begin{bmatrix} 0_{n\times n} & I_n \end{bmatrix}$. *At the start of iteration* $j = 0$ *we have*

$$MS^*\bar{U} = \begin{bmatrix} & & & & * & * & * & * \\ & & & & * & * & * & * \\ & & & & * & * & * & * \\ & & & & 0t_4 & 14t_4 & 14t_4 & 2t_4 \end{bmatrix},$$

with $[a_1, a_2, a_3, a_4] = [0, 14, 14, 2]$. *Working over* $\mathbb{Z}/(21)$*, we solve the system*

$$c_1 0 + c_2 14 + c_3 14 + c_4 2 = 1 \bmod 21,$$

to obtain $c = [c_1, c_2, c_3, c_4] = [0, 2, 2, 4]$. *Thus,* C_0 *and* W_0 *are*

$$\begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & 0 \\ & & & & & 1 & & 2 \\ & & & & & & 1 & 2 \\ & & & & & & & 4 \end{bmatrix} \text{ and } \begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & 21 & 0 & -14 & -14 & 1 \end{bmatrix},$$

respectively. After updating $\bar{U} = \operatorname{rmod}(\bar{U}C_0W_0, S)$ *we have*

$$\bar{U} = \begin{bmatrix} & & & 0 & 0 & 0 & 0 & 0 \\ & & & 0 & 0 & 0 & 2 & 2 \\ & & & 12 & 0 & 2 & 3 & 2 \\ & & & 84 & 0 & 49 & 49 & 4 \end{bmatrix}.$$

Now we move on to iteration $j = 1$. *We have*

$$MS^*\bar{U} = \begin{bmatrix} & & & * & * & * & * & * \\ & & & * & * & * & * & * \\ & & & 6t_3 & 0t_3 & 6t_3 & 13t_3 & * \\ & & & & & & & t_4 \end{bmatrix}$$

with $[a_1, a_2, a_3, a_4] = [6, 0, 6, 13]$. *Working over* $\mathbb{Z}/(15)$, *we solve the system*

$$c_1 6 + c_2 0 + c_3 6 + c_4 13 = 1 \bmod 15$$

to obtain $c = [c_1, c_2, c_3, c_4] = [-1, 0, -1, 1]$. *Thus,* C_1 *and* W_1 *are*

$$\begin{bmatrix} 1 &&&&&&& \\ & 1 &&&&&& \\ && 1 &&&&& \\ &&& 1 &&& -1 & \\ &&&& 1 && 0 & \\ &&&&& 1 & -1 & \\ &&&&&& 1 & \\ &&&&&&& 1 \end{bmatrix} \text{ and } \begin{bmatrix} 1 &&&&&&& \\ & 1 &&&&&& \\ && 1 &&&&& \\ &&& 1 &&&& \\ &&&& 1 &&& \\ &&&&& 1 && \\ && 15 & -6 & 0 & -6 & 1 & \\ &&&&&&& 1 \end{bmatrix}$$

respectively. After updating $\bar{U} = \mathrm{rmod}(\bar{U}C_1W_1, S)$ *we have*

$$\bar{U} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 3 & 0 & 8 & 4 & 2 \\ 0 & 63 & 0 & 28 & 21 & 4 \end{bmatrix}.$$

Now we move on to iteration $j = 2$. *We have*

$$MS^*U = \begin{bmatrix} * & * & * & * & * & * \\ 0 & 9t_2 & 0t_2 & 4t_2 & * & * \\ &&&& t_3 & * \\ &&&&& t_4 \end{bmatrix}$$

with $[a_1, a_2, a_3, a_4] = [0, 9, 0, 4]$. *Working over* $\mathbb{Z}/(15)$ *we solve the equation*

$$c_1 0 + c_2 9 + c_3 0 + c_4 4 = 1 \bmod 15$$

to obtain $[c_1, c_2, c_3, c_4] = [0, 1, 0, -2]$. *Thus,* C_2 *and* W_2 *are*

$$\begin{bmatrix} 1 &&&&&&& \\ & 1 &&&&&& \\ && 1 &&& 0 && \\ &&& 1 && 1 && \\ &&&& 1 & 0 && \\ &&&&& -2 && \\ &&&&&& 1 & \\ &&&&&&& 1 \end{bmatrix} \text{ and } \begin{bmatrix} 1 &&&&&&& \\ & 1 &&&&&& \\ && 1 &&&&& \\ &&& 1 &&&& \\ &&&& 1 &&& \\ & 15 & 0 & -9 & 0 & 1 && \\ &&&&&& 1 & \\ &&&&&&& 1 \end{bmatrix}$$

respectively. After updating $\bar{U} = \mathrm{rmod}(\bar{U}C_2W_2, S)$ *we have*

$$\bar{U} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 2 & 4 & 2 \\ 0 & 0 & 0 & 0 & 7 & 21 & 4 \end{bmatrix}.$$

Since $t_1 = 105$ *implies* $C_3 = W_3 = I_{2n}$, *we can stop. If we let* U *be the submatrix of* $\bar{U}$ *comprised of the last* n *columns, then* MS^*U *will be in Howell form.*

10 SCALED MATRIX VECTOR PRODUCT

In order to obtain our softly cubic complexity, we need to show that the key step in our special Howell triangulation algorithm (Figure 4) and in deducing the Hermite from Howell form (Figure 3) can be computed efficiently. We do this by giving an algorithm for the scaled matrix×vector product problem shown in Figure 5.

`ScaledMatVecProd`(M, S, u, h, p)

Input:

(i) A nonsingular Smith form $S = \mathrm{diag}(s_1, \ldots, s_n) \in \mathbb{Z}^{n\times n}$.
Note: Let $s := s_n$ and $S^* := sS^{-1}$.

(ii) $M \in \mathbb{Z}^{n\times n}$ such that $M = \mathbf{c}\mathrm{mod}(M, S)$.

(iii) $u \in \mathbb{Z}^{n\times 1}$ such that $u = \mathbf{r}\mathrm{mod}(u, S)$.

(iv) A divisor $h \in \mathbb{Z}_{\geq 1}$ of s such that $(s/h)^{-1} M S^* u$ is over $\mathbb{Z}$.

(v) An odd prime p such that $p \perp s$ and $\log p \in \Theta(\mathrm{loglog}\det S)$.

Output:

$v = (v_i)_{1\leq i\leq n} \in \mathbb{Z}/(h)^{n\times 1}$ such that

$$\frac{s}{h}\overbrace{\begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}}^{v} \equiv \overbrace{\begin{bmatrix} m_{11} & \cdots & m_{1n} \\ \vdots & \ddots & \vdots \\ m_{n1} & \cdots & m_{nn} \end{bmatrix}}^{M} \overbrace{\begin{bmatrix} \frac{s}{s_1} & & \\ & \ddots & \\ & & \frac{s}{s_n} \end{bmatrix}}^{S^*} \overbrace{\begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}}^{u} \bmod s.$$

Fig. 5. Problem `ScaledMatVecProd`

From Lemma 14, we know that the sum of the bitlengths of the nontrivial columns of M is bounded by $O(\log\det S)$. Since S^*u has entries reduced modulo s, Lemma 16 shows that the matrix×vector product $\mathrm{Rem}(M(S^*u), s)$ can be computed in

$$O(n(\log\det S)(\log s)) \tag{23}$$

bit operations. Dividing $\mathrm{Rem}(M(S^*u), S)$ by s/h gives the output vector v.

However, the cost estimate in (23) is too high for our purposes. Ideally, we would like to replace the $\log s$ factor in (23) with $\log h$. Instead, we are able to obtain the following slightly weaker result.

Theorem 39. *Problem* `ScaledMatVecProd`(M, S, u, h, p) *can be solved in*

$$O(n(\log\det S)(\log h + \mathrm{loglog}\det S) + (\log\det S)^2) \tag{24}$$

bit operations.

In order to simplify the presentation of the algorithm, let

$$m := \begin{bmatrix} m_1 & \cdots & m_n \end{bmatrix}$$

denote a row of M. Our goal then is to compute a scalar $v \in \mathbb{Z}$ such that

$$\frac{s}{h}v \equiv \overbrace{\begin{bmatrix} m_1 & \cdots & m_n \end{bmatrix} \begin{bmatrix} \frac{s}{s_1} & & \\ & \ddots & \\ & & \frac{s}{s_n} \end{bmatrix}}^{A} \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} \bmod s. \tag{25}$$

Afterwards, we simply replace the row vector m in (25) with the matrix M.

We begin with a high level description of the algorithm. The right hand side of (25), if computed over $\mathbb{Z}$ without taking mod$s$, is given by

$$A = \sum_{i=1}^{n} \frac{s}{s_i} m_i u_i. \tag{26}$$

An *a priori* magnitude bound is $A \in O(s^2 \log \det S)$. The formulation in (26) highlights — since we only require an integer congruent to $A \bmod s$ — that the products $m_i u_i$ can be computed modulo s_i since they are scaled by s/s_i. In Subsection 10.1, we show how to replace the scalar products $m_i u_i$ with dot products that give an integer congruent to $m_i u_i$ modulo s_i. This leads to a formula $D \equiv A \bmod s$ but with magnitude bound $D \in O(sh(\log \det S)^2)$. Then in Subsection 10.2, we show how to exploit the fact that (s/h) is a divisor of D, that is, $(h/s)D \in O(h^2(\log \det S)^2)$.

10.1 Precision reduction via partial linearization

Let $X \in \mathbb{Z}_{>1}$ be a positive radix and, for a nonnegative integer k, define

$$\vec{X}^{(k)} := \begin{bmatrix} X^0 \\ X^1 \\ \vdots \\ X^{k-1} \end{bmatrix} \in \mathbb{Z}^{k \times 1}.$$

For $1 \le i \le n$, we let $\vec{m}_i \in \mathbb{Z}_{\ge 0}^{1 \times k_i}$ be the unique vector of coefficients of the X-adic expansion of m_i, that is, $||\vec{m}_i|| < X$ and $m_i = \vec{m}_i \cdot \vec{X}^{(k_i)}$, where

$$k_i := \left\lceil \frac{\log s_i}{\log X} \right\rceil.$$

We can then rewrite the formula for A in (26) as

$$\begin{aligned} A &= \sum_{i=1}^{n} \frac{s}{s_i} \vec{m}_i \vec{X}^{(k_i)} u_i \\ &= \overbrace{\begin{bmatrix} \vec{m}_1 & \cdots & \vec{m}_n \end{bmatrix}}^{\vec{m}} \begin{bmatrix} \frac{s}{s_1} I_{k_1} & & \\ & \ddots & \\ & & \frac{s}{s_n} I_{k_n} \end{bmatrix} \begin{bmatrix} u_1 \vec{X}^{(k_1)} \\ \vdots \\ u_n \vec{X}^{(k_n)} \end{bmatrix}. \end{aligned} \tag{27}$$

Example 40. *Let $m = [9, 7926]$, $u = [1012, 8057]^t$ and $X = 10$. Then, $A = mu$ can be computed as*

$$A = \overbrace{\left[\begin{array}{c|cccc} 9 & 6 & 2 & 9 & 7 \end{array}\right]}^{[\vec{m}_1, \vec{m}_2]} \overbrace{\left[\begin{array}{c} 1012 \\ \hline 8057 \\ 80570 \\ 805700 \\ 8057000 \end{array}\right]}^{[u_1 \vec{X}^{(1)}, u_2 \vec{X}^{(4)}]^t} = 63868890.$$

For the components of $\vec{m}$, we will often separately consider cases $k_i = 1$ and $k_i > 1$. Note that, in the latter case, $k_i > 1$ implies $(\log s_i)/(\log X) > 1$, and hence we have the upper bound

$$k_i = \left\lceil \frac{\log s_i}{\log X} \right\rceil \le 1 + \frac{\log s_i}{\log X} \le \frac{2 \log s_i}{\log X}. \tag{28}$$

Lemma 41. *The sum of the bitlengths of the entries of $\vec{m}$ is bounded by $O(\log \det S)$.*

Proof. If $k_i = 1$ then $\vec{m}_i$ consists of a single entry bounded in magnitude by $s_i > 1$. The sum of the bitlengths of all such entries of $\vec{m}$ is bounded by

$$\sum_{\substack{i=1\\k_i=1}}^{n} \lg s_i \le \sum_{\substack{i=1\\k_i=1}}^{n} (1+\log s_i) \le \sum_{\substack{i=1\\k_i=1}}^{n} (2\log s_i) \le 2\log\det S.$$

If $k_i > 1$ then $\vec{m}_i$ contains k_i entries with magnitude bounded by X, and thus the sum of the bitlength of entries in $\vec{m}_i$ is bounded by

$$k_i \lg X \le \left(\frac{2\log s_i}{\log X}\right)(1+\log X) \le \left(\frac{2\log s_i}{\log X}\right)(2\log X) \le 4(\log s_i), \tag{29}$$

with the first inequality coming from bound (28). The sum of the right hand side of (29) over all i with $k_i > 1$ is thus also $O(\log \det S)$. □

Now we return to the reformulation of A shown in (27). Since we only require $A \bmod s$, we can preemptively reduce the column vector in (27) by defining $\vec{u}_i := \mathrm{Rem}(u_i\vec{X}^{(k_i)}, s_i)$ for $1 \le i \le n$. Then

$$D := \sum_{i=1}^{n} \frac{s}{s_i}\vec{m}_i\vec{u}_i \tag{30}$$

is congruent to $A \bmod s$.

Example 42. *Let $m = [9, 7926]$, $u = [1012, 8057]^t$ and $X = 10$ be as in Example 40 and set $s = 10000$. Then,*

$$D = \overbrace{\left[\begin{array}{c|cccc} 9 & 6 & 2 & 9 & 7 \end{array}\right]}^{\vec{m}} \overbrace{\left[\begin{array}{c} 1012 \\ \hline 8057 \\ 570 \\ 5700 \\ 7000 \end{array}\right]}^{\vec{u}} = 158890$$

is congruent modulo s to $A = mu$.

Our first lemma derives a bound on the magnitude of D.

Lemma 43. *Let D be defined as in (30). Then $D < 2sX \log \det S$.*

Proof. From (30), we see that

$$D = \left[\begin{array}{ccc} \vec{m}_1 & \cdots & \vec{m}_n \end{array}\right] \left[\begin{array}{c} (s/s_1)\vec{u}_1 \\ \vdots \\ (s/s_n)\vec{u}_n \end{array}\right]$$

is a dot product of length $\sum_{i=1}^{n} k_i$, where the row vector has entries from $[0, X)$, and the column vector has entries from $[0, s)$. This implies $D < sX \sum_{i=1}^{n} k_i$. We can then bound this by

$$\begin{aligned} D &< sX \sum_{i=1}^{n} k_i \\ &= sX \sum_{i=1}^{n} \lceil \log s_i / \log X \rceil \\ &\leq sX \sum_{\substack{i=1 \\ s_i \neq 1}}^{n} (1 + \log s_i) \\ &\leq sX \sum_{\substack{i=1 \\ s_i \neq 1}}^{n} (2 \log s_i) \\ &\leq 2sX \log \det S. \end{aligned}$$

□

Our next lemma bounds the cost of computing vector $\vec{u}_i$. Note that the lemma holds independently of the choice of X (e.g., $X = 2$ is valid).

LEMMA 44. *The vectors $\vec{u}_i$, for $1 \leq i \leq n$, can be computed in $O((\log \det S)^2)$ bit operations.*

PROOF. First consider the cost for a fixed i. If $k_i = 0$, then $\vec{u}_i \in \mathbb{Z}^{0 \times 1}$, and there is no computation needed. Similarly, if $k_i = 1$, then $\vec{u}_i = \begin{bmatrix} u_i \end{bmatrix}$. This leaves us with the case $k_i > 1$. Let

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{k_i} \end{bmatrix} := \vec{u}_i = \begin{bmatrix} \operatorname{Rem}(u_i X^0, s_i) \\ \operatorname{Rem}(u_i X^1, s_i) \\ \vdots \\ \operatorname{Rem}(u_i X^{k_i - 1}, s_i) \end{bmatrix}$$

be our target vector. We can compute the a_i using a Horner scheme by:

$a_1 := u_i$
for $k = 2$ **to** k_i **do**
  $a_k := \operatorname{Rem}(a_{k-1} X, s_i)$
od

By Lemma 15, there exists a constant c such that the cost of one iteration of the loop is bounded by $c(\log X)(\log s_i)$. Since the loop iterates $k_i - 1 < k_i$ times, the total cost to compute $\vec{u}_i$ is $ck_i(\log X)(\log s_i)$. The total cost to compute all $\vec{u}_i$ is then

$$\begin{aligned} \sum_{\substack{i=1 \\ k_i > 1}}^{n} ck_i (\log X)(\log s_i) &\leq c \sum_{\substack{i=1 \\ k_i > 1}}^{n} \left(\frac{2 \log s_i}{\log X} \right) (\log X)(\log s_i) \\ &\leq 2c \sum_{\substack{i=1 \\ k_i > 1}}^{n} (\log s_i)^2 \\ &\leq 2c (\log \det S)^2, \end{aligned}$$

where the first inequality comes from (28). □

10.2 Precision reduction via modular computation

As shown in the proof of Lemma 43, we have

$$D = \begin{bmatrix} \vec{m}_1 & \cdots & \vec{m}_n \end{bmatrix} \begin{bmatrix} (s/s_1)\vec{u}_1 \\ \vdots \\ (s/s_n)\vec{u}_n \end{bmatrix}. \tag{31}$$

In order to reduce the precision of computing this dot product, we can exploit the fact that D has a known divisor s/h, that is, $(h/s)D \in \mathbb{Z}$. Multiplying (31) by (h/s) gives

$$(h/s)D = h \begin{bmatrix} \vec{m}_1 & \cdots & \vec{m}_n \end{bmatrix} \begin{bmatrix} (1/s_1)\vec{u}_1 \\ \vdots \\ (1/s_n)\vec{u}_n \end{bmatrix}. \tag{32}$$

Lemma 43 gives $D < 2sX \log\det S$ and hence $(h/s)D < 2hX \log\det S$. The idea now is to choose a modulus $Y \in \mathbb{Z}_{>0}$ that is relatively prime to s and satisfies $Y \geq 2hX \log\det S$. Then, $(h/s)D < Y$. Since any integer a that satisfies $0 \leq a < Y$ gives $\mathrm{Rem}(a, Y) = a$, we can compute $(h/s)D$ by working modulo Y. To this end, let $\vec{w}_i = \mathrm{Rem}((1/s_i)\vec{u}_i, Y)$ for $1 \leq i \leq n$. Then,

$$(h/s)D = \mathrm{Rem}\left(h \begin{bmatrix} \vec{m}_1 & \vec{m}_2 & \cdots & \vec{m}_n \end{bmatrix} \begin{bmatrix} \vec{w}_1 \\ \vec{w}_2 \\ \vdots \\ \vec{w}_n \end{bmatrix}, Y \right). \tag{33}$$

In order to obtain a good complexity for computing the $\vec{w}_i$ vectors from the $\vec{u}_i$ vectors, the moduli X and Y need to be well chosen.

Lemma 45. *If $X \in \mathbb{Z}_{>0}$ is the smallest power of 2 such that $X > 2h \log\det S$, and $Y \in \mathbb{Z}_{>0}$ is the smallest power of p such that $Y > X^2$, then*

(i) $Y > 2hX \log\det S$,
(ii) $\log Y \in O(\log X)$, and
(iii) the vectors $\vec{w}_i$ for $1 \leq i \leq n$ can be computed from the vectors $\vec{u}_i$ in time $O((\log\det S)^2)$.

Proof. Part (i) follows by substituting $X > 2h \log\det S$ for one of the factors of X in the inequality $Y > X^2$. Part (ii) follows from the choice of Y as the smallest power of p, where $\log p \in O(\mathrm{loglog}\det S)$ as per the problem specification.

For part (iii), we first precompute $\bar{s}_i := \mathrm{Rem}(1/s_i, Y)$ for all $1 \leq i \leq n$. Note that $\bar{s}_i$ can be computed by using the extended euclidean algorithm with input (s_i, Y). Thus, there exists a constant c such that $\bar{s}_i$ can be computed in time $c(\lg s_i)(\lg Y)$. The total cost of computing all the $\bar{s}_i$ is then bounded by

$$\begin{aligned} \sum_{\substack{i=1 \\ s_i \neq 1}}^{n} c(\lg s_i)(\lg Y) &\leq c \sum_{\substack{i=1 \\ s_i \neq 1}}^{n} (1 + \log s_i)(1 + \log Y)) \\ &\leq c \sum_{\substack{i=1 \\ s_i \neq 1}}^{n} (2 \log s_i)(2 \log Y) \\ &\in O((\log\det S)(\log Y)). \end{aligned} \tag{34}$$

The bound (34) is within our target cost since $\log Y \in O(\log h + \mathrm{loglog}\det S)$, which is bounded by $O(\log\det S)$ using the fact that $h \mid \det S$.

Since $\bar{s}_i < Y$ and $||\vec{u}_i|| < s_i$, it follows from Lemma 15 that there exists a constant c' such that the cost of computing $\vec{w}_i := \mathrm{Rem}(\bar{s}_i\vec{u}_i, Y) \in \mathbb{Z}^{k_i\times 1}$ is bounded by $c'k_i(\log s_i)(\log Y)$. To bound the cost of computing all the $\vec{w}_i$ we consider separately the case $k_i = 1$ and $k_i > 1$. For the case $k_i = 1$ we obtain a total cost of

$$\sum_{\substack{i=1\\k_i=1}}^{n} c'(\log s_i)(\log Y) \in O((\log \det S)(\log Y)),$$

which we have already seen to be within our cost bound. For the case $k_i > 1$ we obtain a total cost of

$$\begin{aligned}\sum_{\substack{i=1\\k_i>1}}^{n} c'k_i(\log s_i)(\log Y) &\le c'\sum_{\substack{i=1\\k_i>1}}^{n}\left(\frac{2\log s_i}{\log X}\right)(\log s_i)(\log Y) \ (28)\\ &\le \left(\frac{2c'\log Y}{\log X}\right)\sum_{\substack{i=1\\k_i>1}}^{n}(\log s_i)^2\\ &\le O((\log\det S)^2).\end{aligned}$$

The last inequality uses the fact that $\log Y \in O(\log X)$. □

10.3 Proof of Theorem 39

We first choose dual moduli X and Y as specified in Lemma 45. Construct the partial linearization

$$\vec{M} = \begin{bmatrix} \vec{m}_{11} & \cdots & \vec{m}_{1n}\\ \vdots & \ddots & \vdots\\ \vec{m}_{n1} & \cdots & \vec{m}_{nn}\end{bmatrix} \in \mathbb{Z}^{n\times(k_1+\cdots+k_n)}$$

by replacing column i of M with the $n\times k_i$ matrix containing the coefficients of its X-adic expansion, for $1 \le i \le n$. Since X is a power of 2, the construction of $\vec{M}$ can be done in time linear in the size of M, thus in $O(n \log\det S)$ bit operations.

By Lemmas 44 and 45, we can compute in $O((\log\det S)^2)$ time a vector

$$\vec{w} = \begin{bmatrix}\vec{w}_1\\ \vdots\\ \vec{w}_n\end{bmatrix} \in \mathbb{Z}/(Y)^{(k_1+\cdots+k_n)\times 1} \tag{35}$$

such that our target vector v is then given by $v = \mathrm{Rem}(\mathrm{Rem}(h\vec{M}\vec{w}, Y), h)$. We can thus compute v in three steps:

(1) $a := \mathrm{Rem}(\vec{M}\vec{w}, Y)$
(2) $b := \mathrm{Rem}(ha, Y)$
(3) $v := \mathrm{Rem}(b, h)$

By Lemmas 41 and 16, Step 1 can be done in $O(n(\log\det S)(\log Y))$ bit operations. By Lemma 15, Step 2 has cost $O(n(\log h)(\log Y))$ which, since $h \mid \det S$, is bounded by $O(n(\log\det S)(\log Y))$. Similarly, Step 3 computes n division with remainder operations involving the divisor h and a dividend bounded in magnitude by Y, a step which also has cost $O(n(\log h)(\log Y))$. This shows that once $\vec{w}$ is precomputed, computing the target vector v can be done in time $O(n(\log\det S)(\log Y))$. Finally, by the definition of Y we have that $\log Y \in O(\log h + \mathrm{loglog}\det S)$.

Remark 46. *For clarity, Subsections 10.1 and 10.2 have explained how to construct the vector $\vec{w}$ in (35) in two steps: (a) first construct the vectors $\vec{u}_i \in \mathbb{Z}/(s_i)^{k_i\times 1}$, $1 \le i \le n$, as in Lemma 44; (b) then use Lemma 45 to construct the $\vec{w}_i \in \mathbb{Z}/(Y)^{k_i\times 1}$ from $\vec{u}_i$. An issue with producing $\vec{u}$ explicitly is that it may require $\Omega((\log\det S)^2)$ bits to represent. For this reason, each of the $k_1 + \cdots + k_n$ entries of $\vec{u}$ should be produced one by one and then used to produce the corresponding entry of $\vec{w}$, thus avoiding the need to store $\vec{u}$ explicitly. With this adjustment, the intermediate space requirement of the algorithm remains bounded by $O(n\log\det S)$ bits.*

11 THE HERMITE FORM ALGORITHM

At this point, we have developed all of the components for our algorithm that computes the Hermite form H of a nonsingular integer matrix A.

Before we proceed with our main result, we note that all the algorithms that have been given in Sections 5-10 work with a reduced Smith massager M and a Smith form S, and their cost estimates depend on the dimension n and $\log\det S$. For the Hermite form algorithm, we would like to bound the cost in terms of n and $\log\|A\|$. Since S will be the Smith form of A, by Hadamard's bound, we have that

$$\log\det S = \log|\det A| \le n\log\left(n^{1/2}\|A\|\right). \tag{36}$$

Using (36), the cost estimate $O(n(\log\det S)^2)$ from Theorem 22, directly translates to

$$O(n^3(\log n + \log\|A\|)^2). \tag{37}$$

Similarly, in a slightly less trivial way, the cost estimate

$$O(n(\log\det S)^2 + n^2(\log\det S)(\text{loglog}\det S))$$

from Theorems 37 and 35 is also bounded by (37). The first part is the same as before, and for the second part

$$\begin{aligned} O(n^2(\log\det S)(\text{loglog}\det S)) &\subseteq O(n^3(\log(n\|A\|))(\log(n\log(n\|A\|)))) \\ &\subseteq O(n^3(\log(n\|A\|))(\log n + \log\log(n\|A\|))) \\ &\subseteq O(n^3(\log n + \log\|A\|)^2), \end{aligned}$$

since $O(\log n + \log\log(n\|A\|)) \subseteq O(\log n + \log\|A\|)$.

The following theorem is the main result of the article.

Theorem 47. *There exists a Las Vegas randomized algorithm that computes the Hermite form $H \in \mathbb{Z}^{n\times n}$ of a nonsingular integer matrix $A \in \mathbb{Z}^{n\times n}$. The algorithm uses standard integer and matrix multiplication and has cost $O(n^3(\log n + \log\|A\|)^2(\log n)^2)$ bit operations.*

Proof. The algorithm proceeds in four steps.

(1) $M, S, p :=$ `SmithMassager`(A)
(2) $h_1, \ldots, h_n :=$ `HermiteDiagonals`(A, M, S)
(3) $U :=$ `SpecialHowellTransform`$(A, M, S, [h_1, \ldots, h_n], p)$
(4) $H :=$ `HermiteViaHowell`$(A, M, S, U, [h_1, \ldots, h_n], p)$

Step 1 uses the Las Vegas algorithm of Birmpilis et al. [2020, 2023], restated in Theorem 21, to compute the Smith form S and a reduced Smith massager M of A. The cost is as stated in the current theorem. Note that computing M and S is the only randomized component of the Hermite form algorithm. The Smith massager algorithm also returns a prime p such that $p \perp \det S$. The prime is used in the `ScaledMatVecProd` procedure in the algorithms used in Steps 3 and 4.

Step 2 exploits the fact that M is maintained column modulo S and computes the diagonal entries of H. By Theorem 22 and Hadamard's bound this is done with

$$O(n^3(\log n + \log \|A\|)^2) \tag{38}$$

bit operations.

Step 3 computes a matrix $U \in \mathbb{Z}^{n\times n}$ such that $T = MS^*U$ is right equivalent modulo s to a Howell form of $MS^*(1/s)$, where s is the largest invariant factor in S and $S^* = sS^{-1}$. By Theorem 37 and Hadamard's bound, the time complexity of Step 3 simplifies to (38).

Finally, Step 4 computes the Hermite denominator H of $T(1/s)$. By Theorem 35, the cost of Step 4 is also (38).

To see correctness, note that by Definition 9 the Hermite denominator of $MS^{-1} = MS^*(1/s)$ is the Hermite form of A. Since T is right equivalent to MS^* over $\mathbb{Z}/(s)$, $T(1/s)$ has the same Hermite denominator as $MS^*(1/s)$ (cf. Remark 8). The matrix H computed in Step 4 is thus the Hermite form of A. □

12 USING FAST INTEGER MULTIPLICATION

Our Hermite form algorithm is designed to have a softly cubic complexity in the parameter n in an environment that assumes standard integer multiplication: the cost of multiplying together two integers of bitlength d is $O(d^2)$ bit operations. If we are in an environment where integer multiplication has cost $O(d^{1+\epsilon})$ bit operations for some $0 < \epsilon \leq 1$, we can give a variation of our Hermite form algorithm that establishes the following result.

Theorem 48. *There exists a Las Vegas randomized algorithm that computes the Hermite form $H \in \mathbb{Z}^{n\times n}$ of a nonsingular integer matrix $A \in \mathbb{Z}^{n\times n}$ using $O(n^{3+\epsilon}(\log ||A||)^{1+\epsilon})$ bit operations.*

Before proving the theorem, we give three lemmas. Let $S = \mathrm{diag}(s_1, \ldots, s_n)$ be a nonsingular Smith form, and let $M \in \mathbb{Z}^{n\times n}$ satisfy $M = \mathbf{c}\mathrm{mod}(M, S)$. Also, let $s := s_n$ and $S^* := sS^{-1}$.

Consider the update step $M := \mathbf{c}\mathrm{mod}(H_jM, S)$ required in the proof of Theorem 35. The dominant cost is to compute the outer product of column j of H_j with row j of M, keeping this column reduced modulo S. Our first lemma shows that this can be done efficiently. We also use the lemma in the transpose situation to bound the cost of the update $\bar{U} := \mathbf{r}\mathrm{mod}(\bar{U}W_i, S)$ required in the proof of Theorem 37.

Lemma 49. *Given a $u \in \mathbb{Z}/(s)^{n\times 1}$, together with an $m \in \mathbb{Z}^{1\times n}$ such that $m = \mathbf{c}\mathrm{mod}(m, S)$, we can compute $\mathbf{c}\mathrm{mod}(um, S)$ in $O(n(\log\det S)^{1+\epsilon})$ bit operations.*

Proof. Let $m = \begin{bmatrix} m_1 & \cdots & m_n \end{bmatrix} \in \mathbb{Z}^{1\times n}$. Then

$$\mathbf{c}\mathrm{mod}(um, S) = \begin{bmatrix} \bar{u}_1 & \cdots & \bar{u}_n \end{bmatrix},$$

where $\bar{u}_i = \mathrm{Rem}(um_i, s_i) \in \mathbb{Z}/(s_i)^{n\times 1}$, $1 \leq i \leq n$. Note that if $s_i = 1$ then $\bar{u}_i$ is necessarily the zero vector. The $\bar{u}_*$ that are not necessarily zero can be computed using the following loop:

$\bar{u} := u$
$\bar{u}_n := \mathrm{Rem}(\bar{u}m_n, s_n)$
for i **from** $n-1$ **downto** 1 **do**
  if $s_i = 1$ **then break fi**
  $\bar{u} := \mathrm{Rem}(\bar{u}, s_i)$
  $\bar{u}_i := \mathrm{Rem}(\bar{u}m_i, s_i)$
od

The cost of computing $\bar{u}_n$ is bounded by $O(n(\log s)^{1+\epsilon})$ bit operations. Since the operands at loop iteration i have bitlength bounded by $\lg s_{i+1}$, the cost at iteration i is $O(n(\lg s_{i+1})^{1+\epsilon})$ bit operations. The total cost of the loop is thus $O(n\sum_{i=2,s_i\neq 1}^{n}(\lg s_i)^{1+\epsilon})$. Using the fact that $\sum_{i=2,s_i\neq 1}\lg s_i \in O(\log\det S)$, the total cost to compute the $\bar{u}_*$ is as stated in the lemma. □

Furthermore, consider the update step $\bar{U} := \mathrm{rmod}(\bar{U}C_i, S)$ in the proof of Theorem 37. Since C_i has at most one nontrivial column, the dominant cost is to compute a matrix×vector product, keeping this row reduced modulo S. The following corollary, applied to the transpose situation, shows that this can be done efficiently. The proof is analogous to the proof of Lemma 49.

Corollary 50. *Given a $u \in \mathbb{Z}/(s)^{1\times n}$, together with an $M \in \mathbb{Z}^{n\times n}$ such that $M = \mathrm{cmod}(M, S)$, we can compute $\mathrm{cmod}(uM, S)$ in $O(n(\log\det S)^{1+\epsilon})$ bit operations.*

The following result will be used in place of `ScaledMatVecProd`.

Lemma 51 (Storjohann [2015, Lemma 4.11]). *Given an $M \in \mathbb{Z}^{n\times n}$ such that $M = \mathrm{cmod}(M, S)$, together with a $U \in \mathbb{Z}^{n\times n}$ such that $U = \mathrm{rmod}(U, S)$, then any individual row or column of $\mathrm{Rem}(MS^*U, s)$ can be computed using $O(n(\log\det S)^{1+\epsilon})$ bit operations.*

We now prove Theorem 48.

Proof. (Of Theorem 48). We begin by (i) computing the Smith form S and a reduced Smith massager M of A, then (ii) compute an integer matrix U such that $M(s_nS^{-1})U$ is right equivalent to a Howell form of sA^{-1} over $\mathbb{Z}/(s)$, and finally (iii) compute H as the Hermite denominator of $MS^{-1}U$.

Birmpilis et al. [2023, Theorem 19] establish that phase (i) can be done within the time stated in Theorem 48.

For phase (ii), we adapt the algorithm, with n iterations and three steps per iteration, given in the proof of Theorem 37. In Step 1, use Lemma 51 to compute the required n entries

$$\begin{bmatrix} t_{n-i}a_1 & \cdots & t_{n-i}a_{n-1} & t_{n-i}a_n \end{bmatrix} \in \mathbb{Z}/(s)^{1\times n}. \tag{39}$$

Since we are not given $t_{n-i} = s/h_{n-i}$ as input, we compute it now as the gcd of entries of the n elements in (39) at a cost of

$$O(n(\log s)^{1+\epsilon}) \tag{40}$$

bit operations. In Step 2, the update matrices C_i and W_i can be computed in the time (40) using an analog of Lemma 36. In Step 3, the update $\bar{U} := \mathrm{rmod}(\bar{U}C_iW_i, S)$ is done in time (40) using Lemmas 50 and 49. Since there are n iterations, and $\lg s \le \lg\det S \in O(n(\log n + \log||A||))$, the overall cost of phase (ii) is as stated in the theorem.

Similar to phase (ii), the overall cost bound for phase (iii) follows by adapting the two-step algorithm in the proof of Theorem 35 by using Lemma 51 for Step 1, and Lemma 49 for Step 2. □

Finally, if we assume we are using a pseudo-linear algorithm for integer multiplication, such as the $O(d\log d)$ algorithm of Harvey and van der Hoeven [2021], we obtain the following corollary.

Corollary 52. *There exists a Las Vegas randomized algorithm that computes the Hermite form $H \in \mathbb{Z}^{n\times n}$ of a nonsingular integer matrix $A \in \mathbb{Z}^{n\times n}$ using $(n^3\log||A||)^{1+o(1)}$ bit operations. This cost estimate assumes the use of a pseudo-linear algorithm for integer multiplication.*

13 CONCLUSION AND TOPICS FOR FUTURE RESEARCH

We have given a Las Vegas randomized algorithm to compute the Hermite form $H \in \mathbb{Z}^{n\times n}$ of a nonsingular matrix $A \in \mathbb{Z}^{n\times n}$. The algorithm has worst-case expected running time

$$O(n^3(\log n + \log ||A||)^2(\log n)^2) \tag{41}$$

bit operations using standard integer and matrix multiplication.

The core tool used is the Smith massager which helps control the size of intermediate results. The $(\log n)^2$ factor in (41) is due to the first step of the algorithm, which computes a Smith form S and Smith massager M of A. This first step is accomplished using the Las Vegas algorithm of Birmpilis et al. [2023, Theorem 19] which allows the use of fast matrix multiplication, and shows that S and M can be computed using an expected number of $O(n^\omega (\log n + \log ||A||)^2(\log n)^2)$ bit operations assuming standard integer multiplication. Computing M is also the only part of the Hermite form algorithm that requires randomization.

Once M is precomputed, the algorithm in this paper computes H deterministically using a further $O(n^3(\log n + \log ||A||)^2)$ bit operations. The intermediate space requirement of the algorithm to compute H from M is bounded by $O(n^2(\log n + \log ||A||))$ bits, which is the same as that required to write down H in the worst case.

We have also given a variant of our Hermite form algorithm that has a worst case expected running time $(n^3 \log ||A||)^{1+o(1)}$ bit operations, assuming the use of a pseudo-linear algorithm for integer multiplication.

Our Hermite form algorithms extend to the case of an input matrix $A \in \mathbb{Z}^{m\times n}$ of full column rank n and $m > n$. Up to a row permutation, and up to adding at most $n-1$ zero rows, we may assume without loss of generality that

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_k \end{bmatrix} \in \mathbb{Z}^{kn\times n},$$

where each A_* is $n \times n$, $A_1 \in \mathbb{Z}^{n\times n}$ is nonsingular, and $k = \lceil n/m \rceil$. Initialize $H_1 := A_1$. Compute, in succession for $i = 2, 3, \ldots, k$, the leading principal $n \times n$ submatrix H_i of the Hermite form of the nonsingular matrix

$$\begin{bmatrix} H_{i-1} & \\ A_i & I_n \end{bmatrix} \in \mathbb{Z}^{2n\times 2n}. \tag{42}$$

Then $H_k \in \mathbb{Z}^{n\times n}$ is the leading principal $n \times n$ submatrix of the Hermite form of A. Birmpilis et al. [2023, Theorem 27 and Remark 34] show that computing the Hermite form of (42) reduces to that of computing the Hermite form of a matrix of dimension bounded by $4n$ that has entries with bitlength $O(\log n + \log ||A||)$. Computing the Hermite form of an $A \in \mathbb{Z}^{m\times n}$ of rank n can thus be done in a Las Vegas fashion using an expected number of $O(mn^2(\log n + \log ||A||)^2(\log n)^2)$ bit operations using standard integer and matrix arithmetic, or an expected number of $O(mn^2 \log ||A||)^{1+o(1)}$ bit operations using pseudo-linear integer multiplication.

In terms of future directions, a natural goal is to find an algorithm to compute the Hermite form of a nonsingular integer matrices that has cost $(n^\omega \log ||A||)^{1+o(1)}$ bit operations. In addition, we would like to find a deterministic algorithm for the Hermite form problem with the same complexity.

REFERENCES

J. Alman and V. V. Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539, 2021. doi: 10.1137/1.9781611976465.32.

E. Bach and J. Shallit. *Algorithmic Number Theory*, volume 1 : Efficient Algorithms. MIT Press, 1996.

B. Beckermann and G. Labahn. A uniform approach for the fast computation of matrix–type Padé approximants. *SIAM Journal on Matrix Analysis and Applications*, 15(3):804–823, 1994.

B. Beckermann, G. Labahn, and G. Villard. Shifted normal forms of polynomial matrices. In S. Dooley, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'99*, pages 189—196. ACM Press, New York, 1999.

S. Birmpilis, G. Labahn, and A. Storjohann. A Las Vegas algorithm for computing the Smith form of a nonsingular integer matrix. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'20*, page 38–45, New York, NY, USA, 2020. ACM.

S. Birmpilis, G. Labahn, and A. Storjohann. A fast algorithm for computing the Smith normal form with multipliers for a nonsingular integer matrix. *Journal of Symbolic Computation*, 116:146–182, 2023.

T-W. J. Chou and G. E. Collins. Algorithms for the solutions of systems of linear diophantine equations. *SIAM Journal of Computing*, 11:687–708, 1982.

G. E. Collins. Computing time analyses for some arithmetic and algebraic algorithms. Technical Report 36, University of Wisconsin, Madison; Computer Sciences, July 1968.

P. D. Domich, R. Kannan, and L. E. Trotter, Jr. Hermite normal form computation using modulo determinant arithmetic. *Mathematics of Operations Research*, 12(1):50–59, 1987.

J. L. Hafner and K. S. McCurley. A rigorous subexponential algorithm for computation of class groups. *J. Amer. Math. Soc.*, 2:837–850, 1989.

D. Harvey and J. van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193:563–617, 2021.

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

J. A. Howell. Spans in the module $(\mathbb{Z}_m)^s$. *Linear and Multilinear Algebra*, 19:67—77, 1986.

E. Hubert and G. Labahn. Scaling invariants and symmetry reduction of dynamical systems. *Foundations of Computational Mathematics*, 13(4):479–516, 2013.

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 Journal of Computing*, 18(4):658–669, 1989.

R. Kannan and A. Bachem. Polynomial algorithms for computing the Smith and Hermite normal forms of and integer matrix. *SIAM Journal of Computing*, 8(4):499–507, November 1979.

G. Labahn, V. Neiger, and W. Zhou. Fast, deterministic computation of the Hermite normal form and determinant of a polynomial matrix. *J. Complexity*, 42:44–71, 2017.

R. Liu and Y. Pan. Computing Hermite normal form faster via solving system of linear equations. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'19*, page 283–290, New York, NY, USA, 2019. ACM.

D. Micciancio and B. Warinschi. A linear space algorithm for computing the Hermite normal form. In B. Mourrain, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'01*, pages 231—236. ACM Press, New York, 2001.

C. Pauderis and A. Storjohann. Computing the invariant structure of integer matrices: Fast algorithms into practice. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'13*, pages 307–314, New York, NY, USA, 2013. ACM.

C. Pernet and W. Stein. Fast computation of Hermite normal forms of integer matrices. *Journal of Number Theory*, 130(7): 1675–1683, 2010.

A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, 1998.

A. Storjohann. *Algorithms for Matrix Canonical Forms*. PhD thesis, Swiss Federal Institute of Technology, ETH–Zurich, 2000.

A. Storjohann. On the complexity of inverting integer and polynomial matrices. *Computational Complexity*, 24(4):777—821, 2015.

A. Storjohann and G. Labahn. Asymptotically fast computation of Hermite normal forms of integer matrices. In Y. N. Lakshman, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'96*, pages 259–266. ACM Press, New York, 1996.

A. Storjohann and T. Mulders. Fast algorithms for linear algebra modulo N. In G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, editors, *Algorithms — ESA '98*, LNCS 1461, pages 139–150. Springer Verlag, 1998.

W. Zhou and G. Labahn. Efficient algorithms for order basis computation. *Journal of Symbolic Computation*, 47(7):793–819, 2012.

W. Zhou and G. Labahn. Computing column bases of polynomial matrices. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'13*, pages 379–388. ACM Press, Boston, USA, 2013.

W. Zhou, G. Labahn, and A. Storjohann. Computing minimal nullspace basis. In J. van der Hoeven and M. van Hoeij, editors, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'12*, pages 366–373. ACM Press, New York, 2012.