A Heuristic Algorithm for Integer Hermite Normal Form

by

Xiaoyu Liu

A research paper
presented to the University of Waterloo
in partial fulfillment of the
requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Supervisor: Prof. Arne Storjohann

Waterloo, Ontario, Canada, 2017

I hereby declare that I am the sole author of this report. This is a true copy of the report, including any required final revisions, as accepted by my examiners.

I understand that my report may be made electronically available to the public.

Abstract

This report describes a new heuristic algorithm to compute the upper triangular (row) Hermite normal form of an integer matrix $A \in \mathbb{Z}^{n \times m}$ that has full column rank. The algorithm has three features. First, the algorithm is online: column k of A can be given one at a time for $k = 1, 2, \ldots, m$. As soon as the first k columns of A are known the algorithm will produce column k of the Hermite form. Second, the algorithm has a running time that, in terms of n and m, seems to be within a polylogarithmic factor of $O(nm^2)$ bit operations. Assuming standard matrix multiplication, this is a factor of about m faster than previous algorithms that are deterministic and analysed in the worst case. Third, the intermediate space requirements of the algorithm seem to be, in terms of bits, about the same as the number of bits required to write down a dense input matrix. Empirical results from a Maple implemention of the algorithm are discussed.

Acknowledgements

I would like to thank to my supervisor, Prof. Arne Storjohann, for all the guidance, support, patience and encouragement. I would also like to thank Prof. George Labahn for his feedback on this project.

Dedication

To my parents for their unconditional commitment to support me.

To my boyfriend, Shenghao, for providing much motivation and inspiration, without his love this work would be meaningless to do.

Table of Contents

Chapter 1

Introduction

In elementary linear algebra, a common computation is to transform an input matrix to a canonical form. One of the more useful canonical forms of an integer matrix is the Hermite form. Let $A \in \mathbb{Z}^{n \times m}$ have full column rank m. The (row) Hermite form of A is an upper triangular matrix that is left equivalent to A. By *Hermite basis* we mean the submatrix of the Hermite form comprised of the nonzero rows. The Hermite basis of A is then

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

where the off-diagonal entries h_{*i} satisfy $0 \leq h_{*i} < h_i$ for $i = 1, 2, \ldots, m$. These conditions on the h_{*i} ensure uniqueness of the form.

For example, the Hermite form of

$$A = \begin{bmatrix} 4 & 8 & 3 \\ 9 & 10 & 2 \\ 8 & 10 & 9 \end{bmatrix},$$

is

$$H = \begin{bmatrix} 1 & 0 & 98 \\ & 2 & 34 \\ & & 105 \end{bmatrix}.$$

A unimodular matrix U satisfying $H = UA$ is a transformation matrix with a sequence of integer row operations for A. The usual method to compute the Hermite form is to perform a sequence of elementary row operations. These are

- interchanging two rows
- adding an integer multiple of one row to another
- negating a row

By recording these row operations a unimodular matrix $U \in \mathbb{Z}^{n\times n}$ such that $UA = H$ can be constructed.

The computation of the Hermite normal form is a vital tool for many algebra problems. For instance, it can accelerate linear system solving, be used to check the span of a matrix, and assist in integer program solving [Hung, 1990]. In more applicable areas, Hermite normal form is suggested to improve the security and efficiency of lattice based cryptography by reducing the size of a public key [Micciancio, 2001].

The running time and space requirements are two primary measurements of the efficiency of an algorithm. Over several decades many people worked on deriving or optimizing algorithms to compute the Hermite normal form. The first polynomial bounded algorithm was developed by Kannan and Bachem [1979]. They only preformed basic row operations and computed principal minors of a matrix to derive the Hermite normal form. Their bound on the size of the entries was improved by Chou and Collins [1982]. They normalized the entries above the main diagonal and achieved better bounds for both cost and storage. Hafner and McCurley [1989] showed how to incorporate matrix multiplication to triangularize an integer matrix. Storjohann and Labahn [1996] successfully used fast matrix multiplication to speed up Hermite normal form computation.

Table 1.1 gives a history of results in the past several decades. It summarizes polynomial time complexity results for the case of a nonsingular $n \times n$ input matrix A. The Time and Space are expressed in terms of an exponent e such that required number of bit operations and intermediate space requirements (in bits) is bounded by $O\left(n^e \log ||A||\right)^*)$ with corresponding value of e shown in table, where for most algorithms $*$ is a small number, typically 1. Here $||A||$ denotes the maximum in absolute value of the entries of A. In Table 1.1, the last two algorithms differ from the previous in two senses: they are randomized, and the time and space complexity is not claimed to be worst case, but rather indicative on the running time for most inputs.

The rest of this report is organized as follows. Chapter 2 describes a previous algorithm to compute the Hermite normal form via Gaussian elimination. Chapter 3 characterizes two refinements of the algorithm to factor out the Hermite basis and to utilize lattice compression to improve the running time. A key subroutine in our algorithm is to apply the outer product adjoint to solve linear system and to update the Hermite basis; this is

Table 1.1: Algorithms for Computing the Hermite Form

Citation	Time	Space
Kannan and Bachem [1979]	6	3
Chou and Collins [1982]	4	3
Domich [1985]	4	3
Domich et al. [1987]	4	3
Iliopoulos [1989]	4	3
Hafner and McCurley [1989]	4	3
Storjohann and Labahn [1996]	$\theta + 1$	3
Storjohann [2000]	4	2
Pauderis and Storjohann [2012]	3	2
This report	3	2

described in Chapter 4. Finally, Chapter 5 concludes by summarizing the entire algorithm and presents some empirical results of a Maple implementation.

Chapter 2

Hermite Form via Gaussian Elimination

In this chapter we recall the algorithm of Storjohann [1996, 2003] which computes the Hermite form via Gaussian elimination. The algorithm makes use of the modulo extended gcd algorithm to control the growth of integers in the matrix being transformed: the integers in the work matrix grow in bitlength similarly as if fraction free Gaussian elimination is performed.

Let T be a copy of an input matrix $A \in \mathbb{Z}^{n\times m}$ that has full column rank m. The algorithm directly performs unimodular row operations on T to transform T to Hermite form, proceeding in stages for column $k = 1, 2, \ldots, m$. The entire algorithm can be understood by considering a single stage k. Let T_k be the state of the work matrix at the start of stage k. Then T_k has the shape

$$T_k = \left[\begin{array}{cccc|c|ccc}
h_1 & \cdots & h_{k-1\,1} & * & * & * & \cdots & * \\
 & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
 & & h_{k-1} & * & * & * & \cdots & * \\
 & & & d & \bar{d} & * & \cdots & * \\
\hline
 & & & a & \bar{a} & * & \cdots & * \\
 & & & b_1 & \bar{b}_1 & * & \cdots & * \\
 & & & \vdots & \vdots & \vdots & \ddots & \vdots \\
 & & & b_* & \bar{b}_* & * & \cdots & *
\end{array}\right] \in \mathbb{Z}^{n\times m},$$

where the principal $k \times k$ submatrix is in Hermite form, d is nonzero, and entries below d in column k are reduced modulo d. The goal at stage k is to transform the first k columns to Hermite form. The key idea of the algorithm is to precondition row $k+1$ of T_k by adding small integer multiples of rows $k+2, k+2, \ldots, m$ to row $k+1$. The modulo d extended gcd algorithm will efficiently compute the lexicographically minimal sequence of nonnegative integers $c_1, c_2, \ldots, c_*$ such that $\gcd(d, a, b_1, \ldots, b_*) = \gcd(d, a + c_1 b_1 + \ldots c_* b_*)$, where $* = n-k-1$. Once the c_* have been computed, form the unimodular preconditioning matrix

$$C_k := \left[\begin{array}{cccc|cccc} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ \hline & & & & 1 & c_1 & \cdots & c_* \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{array}\right] \in \mathbb{Z}^{n \times n}.$$

Then

$$C_k T_k = \left[\begin{array}{cccc|c|ccc} h_1 & \cdots & h_{k-1\,1} & * & * & * & \cdots & * \\ & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ & & h_{k-1} & * & * & * & \cdots & * \\ & & & d & \bar{d} & * & \cdots & * \\ \hline & & & \ell & \bar{\ell} & * & \cdots & * \\ & & & b_1 & \bar{b}_1 & * & \cdots & * \\ & & & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & b_* & \bar{b}_* & * & \cdots & * \end{array}\right],$$

where $l = a + c_1 b_1 + \cdots + c_* b_*$ and thus $\gcd(d, l) = \gcd(d, a, b_1, ..., b_*)$, which is h_k by definition. We remark that during the computation of the c_i's we also ensure that the 2×2 minor

$$\left| \begin{array}{cc} d & \bar{d} \\ l & \bar{l} \end{array} \right| = d\bar{l} - \bar{d}l$$

is nonzero. Next use the extended Euclidean algorithm to compute the Bezout matrix

$$\left[\begin{array}{cc} s & t \\ u & v \end{array} \right] \in \mathbb{Z}^{2\times 2}$$

such that

$$\left[\begin{array}{cc} s & t \\ u & v \end{array} \right] \left[\begin{array}{cc} d & \bar{d} \\ l & \bar{l} \end{array} \right] = \left[\begin{array}{cc} h_k & * \\ & e \end{array} \right] \in \mathbb{Z}^{2\times 2}$$

is in Hermite form. It is easy to see how to extend this Bezout matrix to the unique $n \times n$ unimodular matrix

$$Q_k = \left[\begin{array}{c|cc|c} I_{k-1} & * & * & \\ \hline & s & t & \\ & u & v & \\ \hline & * & * & I_{n-k-1} \end{array}\right] \in \mathbb{Z}^{n\times n}$$

such that

$$T_{k+1} := Q_k C_k T_k = \left[\begin{array}{ccccc|ccc} h_1 & \cdots & h_{k-1\,1} & h_{k\,1} & * & * & \cdots & * \\ & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ & & h_{k-1} & h_{k\,k-1} & * & * & \cdots & * \\ & & & h_k & * & * & \cdots & * \\ & & & & e & * & \cdots & * \\ \hline & & & & * & * & \cdots & * \\ & & & & \vdots & \vdots & \vdots & \vdots \\ & & & & * & * & \ddots & * \end{array}\right],$$

that is, such that the principal $(k+1) \times (k+1)$ submatrix is nonsingular and in Hermite form, and that entries below e are reduced modulo e. This completes the description of stage k of the algorithm.

In Storjohann [2003] it is shown that the bitlengths of integers in T_k will be bounded by $O(k(\log k + \log |A||))$ bits. Not counting the time for the calls to the modulo extended gcd algorithm, which both in theory and in practice is negligible, the overall running time of this algorithm is thus $O(nm^2)$ operations on integers with bitlength bounded by $O(m(\log m + \log ||A||))$, or exactly the same as fraction free Gaussian elimination. A serious issue with the approach is that the space requirements are relatively high because at stage k the matrix T_k has last $n-k$ columns filled with large integers. Actually, in Storjohann [2003] an online version of the algorithm is presented that computes column $k+1$ of T_k at the start of stage k. We will describe this online version in Section 2.1, but first we give a worked example.

Example 1. *Let*

$$A = \begin{bmatrix} -175 & -105 & 5 & -2 \\ -40 & 140 & 2 & 118 \\ -94 & -70 & -68 & 82 \\ -23 & -35 & -28 & -81 \\ -174 & 70 & 78 & 104 \\ 30 & 0 & -31 & -151 \\ 76 & 70 & 25 & 11 \end{bmatrix} \in \mathbb{Z}^{7\times 4}.$$

At stage $k = 1$, the preconditioning matrix C_1 is used to transform A to ensure that the gcd of the first two entries in column 1 are equal to the gcd of all entries in column 1.

$$
C_1 A = \left[\begin{array}{cc|ccccc} 1 & & & & & & \\ & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline & & 1 & & & & \\ & & & 1 & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & & & & 1 \end{array}\right] \begin{bmatrix} -175 & -105 & 5 & -2 \\ -40 & 140 & 2 & 118 \\ -94 & -70 & -68 & 82 \\ -23 & -35 & -28 & -81 \\ -174 & 70 & 78 & 104 \\ 30 & 0 & -31 & -151 \\ 76 & 70 & 25 & 11 \end{bmatrix}
$$

$$
= \begin{bmatrix} -175 & -105 & 5 & -2 \\ -134 & 70 & -66 & 200 \\ -94 & -70 & -68 & 82 \\ -23 & -35 & -28 & -81 \\ -174 & 70 & 78 & 104 \\ 30 & 0 & -31 & -151 \\ 76 & 70 & 25 & 11 \end{bmatrix}.
$$

Now we apply the extended Bezout matrix Q_1 (not shown) to obtain

$$
T_2 = Q_1 C_1 A = \begin{bmatrix} 1 & 16695 & -7751 & 22370 \\ & 26320 & -12220 & 35268 \\ & 16380 & -7682 & 22050 \\ & 15470 & -7221 & 20677 \\ & 9800 & -4396 & 13004 \\ & -770 & 319 & -1159 \\ & -5390 & 2541 & -7245 \end{bmatrix}.
$$

Stage $k = 2$ uses C_2 to precondition the second column of T_2 to obtain

$$C_2T_2 = \left[\begin{array}{ccc|cccc} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & 1 & 0 & 0 & 0 \\ \hline & & & 1 & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & & & & 1 \end{array}\right] \begin{bmatrix} 1 & 16695 & -7751 & 22370 \\ & 26320 & -12220 & 35268 \\ & 16380 & -7682 & 22050 \\ & 15470 & -7221 & 20677 \\ & 9800 & -4396 & 13004 \\ & -770 & 319 & -1159 \\ & -5390 & 2541 & -7245 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 16695 & -7751 & 22370 \\ & 26320 & -12220 & 35268 \\ & 31850 & -14903 & 42727 \\ & 15470 & -7221 & 20677 \\ & 9800 & -4396 & 13004 \\ & -770 & 319 & -1159 \\ & -5390 & 2541 & -7245 \end{bmatrix},$$

and then applies Q_2 to obtain

$$T_3 = Q_2C_2T_2 = \begin{bmatrix} 1 & 35 & 14075 & -5928 \\ & 70 & 29651 & -12491 \\ & & 43428 & -18412 \\ & & -2464 & 976 \\ & & -29876 & 12604 \\ & & 22484 & -9676 \\ & & 27412 & -11628 \end{bmatrix}.$$

At stage $k = 3$, $\gcd(43428, -2464) = h_3$, so C_3 is just the identity matrix, and we need only apply Q_3 to transform the third column into the correct form:

$$T_4 = Q_3T_3 = \begin{bmatrix} 1 & 35 & 215 & 772 \\ & 70 & 83 & 8901 \\ & & 308 & 3508 \\ & & & 9680 \\ & & & 4400 \\ & & & -4400 \\ & & & -4400 \end{bmatrix}.$$

At stage 4, C_4 is again the identity matrix. Finally we obtain

$$Q_4T_4 = \begin{bmatrix} 1 & 35 & 215 & 772 \\ & 70 & 83 & 101 \\ & & 308 & 868 \\ & & & 880 \\ & & & \\ & & & \\ & & & \end{bmatrix},$$

the Hermite form of A.

2.1 An On-line Variation

From the description above, it should be clear that only the first $k+1$ columns of T_k are required to obtain the first $k+1$ columns of T_{k+1}. In particular, the matrices C_k and Q_k depend only on the first $k+1$ columns of T_k. The idea of the on-line algorithm is to compute column $k+1$ of T_k when it is needed, at that start of stage k. In the online algorithm we apply the preconditioning matrices directly to the input matrix A so that at stage k we have

$$A_k \quad = \quad C_{k-1} \cdots C_2 C_1 A.$$

At the start of stage k, we thus can write the preconditioned matrix A_k using a block decomposition as

$$A_k = \left[\begin{array}{c|c||c} B & b & \cdots \\ \hline F & f & \cdots \end{array} \right] \in \mathbb{Z}^{n \times (k+1)},$$

where B is $k \times k$ and nonsingular, $f \in \mathbb{Z}^{k \times 1}$, and entries to the right of the double vertical line do not even need to be known at this point. Moreover, the Hermite basis of the first $k-1$ columns of B is the Hermite basis of the first $k-1$ columns of A. The matrix T_k has the shape

$$T_k = \left[\begin{array}{cccc|c||c} h_1 & \cdots & h_{k-1\,1} & * & * & \cdots \\ & \ddots & \vdots & \vdots & \vdots & \cdots \\ & & h_{k-1} & * & * & \cdot \\ & & & d & \bar{d} & \cdots \\ \hline & & & a & \bar{a} & \cdots \\ & & & b_1 & \bar{b}_1 & \cdots \\ & & & \vdots & \vdots & \cdots \\ & & & b_* & \bar{b}_* & \cdots \end{array}\right] \in \mathbb{Z}^{n\times m}$$

where entries to the right of the double vertical line do not need to be known. Actually, in the on-line algorithm, at the start of stage k we only have the first k columns of T_k, namely

$$\left[\begin{array}{c} H \\ \hline C \end{array}\right] := \left[\begin{array}{cccc} h_1 & \cdots & h_{k-1\,1} & * \\ & \ddots & \vdots & \vdots \\ & & h_{k-1} & * \\ & & & d \\ \hline & & & a \\ & & & b_1 \\ & & & \vdots \\ & & & b_* \end{array}\right] \in \mathbb{Z}^{n\times k}.$$

A main step in the on-line algorithm is to compute column $k+1$ of T_k, that is,

$$\left[\begin{array}{c} \bar{h} \\ \bar{d} \\ \hline \bar{a} \\ \bar{b}_1 \\ \vdots \\ \bar{b}_* \end{array}\right] \in \mathbb{Z}^{n\times(k+1)}.$$

It is easy to deduce that this column is given by

$$\overbrace{\left[\begin{array}{c|c} H & \\ \hline C & I_{n-k} \end{array}\right]\left[\begin{array}{c|c} B^{-1} & \\ \hline -FB^{-1} & I_{n-k} \end{array}\right]}^{U}\left[\begin{array}{c} b \\ \hline f \end{array}\right]. \tag{2.1}$$

where $U \in \mathbb{Z}^{n\times n}$ is a unimodular transforming matrix such that $UA_k = T_k$. Because A_k is preconditioned there exists a unique such unimodular transform matrix that has last $n-k$ columns that of the I_n, namely the U shown in (2.1).

One of the main computations involved in (2.1) is to compute $HB^{-1}b$. Since H is triangular we may easily deduce its determinant. We can now structure the computation of $HB^{-1}b$ as

$$(1/(\det H))H(B^{-1}((\det H)b)),$$

where the nonsingular rational system solution $B^{-1}((\det H)b)$ is integral. An option to compute this system solution is to use p-adic lifting. Computing the lower part of (2.1) is similar. The overall cost of this online version is shown to be only $O(nm^3(\log m + \log ||A||)^2)$ bit operations, even assuming standard integer arithmetic. Moreover, the intermediate space requirements are reduced to $O(nm(\log m + \log ||A||))$ bits, or about the same as required to write down the input matrix. We refer to Storjohann [2003] for more details.

The online algorithm can thus be summarized as follows. To begin, assume without loss of generality, up to some elementary row operations, that A_{11} is positive, and define the first column of T_1 to be the first column of A. Now, for $k = 1, 2, \ldots, m-1$ do the following steps:

1. Compute column $k+1$ of T_k as described above.
2. Compute C_k and Q_k from T_k.
3. Define the first $k+1$ columns of T_{k+1} to be those of $Q_k C_k T_k$.
4. Let $A_{k+1} = C_k A_k$.

At the end of stage $m-1$ we have computed

$$T_m = \left[\begin{array}{cccc} h_1 & \cdots & h_{m-1\,1} & * \\ & \ddots & \vdots & \vdots \\ & & h_{m-1} & * \\ & & & d \\ \hline & & & a \\ & & & b_1 \\ & & & \vdots \\ & & & b_* \end{array}\right],$$

from which the Hermite form of A can be easily recovered using some additional operations on only the last column, in particular, computing $h_m = \gcd(d, a, b_1, \ldots, b_*)$ and reducing the $*$ entries modulo h_m.

Chapter 3

Our Refinement of the Algorithm

In this chapter we explain two refinements of the online algorithm from Section 2.1. Our first refinement is based on the observation that, if $A \in \mathbb{Z}^{n\times m}$ has full column rank and the Hermite basis of A is $H \times \mathbb{Z}^{m\times m}$, then AH^{-1} is also integral, and in fact has Hermite basis equal to I_m. In Section 3.1 we show how the Hermite basis can be gradually factored out of A, column by column for $k = 1, 2, \ldots, m$. This simplifies the process of going from stage k to $k+1$. In Section 3.2 we show how to apply a lattice compression technique to avoid computing most of the entries in the T_k matrix at stage k.

3.1 Factoring out the Hermite basis

Let the Hermite basis of our input matrix $A \in \mathbb{Z}^{n\times m}$ be

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

Note that

$$H = H_m \cdots H_2 H_1$$

where

$$H_k = \begin{bmatrix} 1 & & & h_{1,k} & & & \\ & \ddots & & \vdots & & & \\ & & 1 & h_{k-1,k} & & & \\ & & & h_k & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix} \in \mathbb{Z}^{m\times m}.$$

For any $0 \le k \le m$, the matrix $AH_1^{-1}H_2^{-1}\cdots H_k^{-1}$ will be integral and the Hermite basis of the first k colums will be I_k. The first refinement of the algorithm is, at stage k, to compute H_{k+1} and remove this factor from the input matrix in preperation for stage $k+1$. So, at the start of stage k we have the matrix $A_k = C_{k-1}\cdots C_2C_1AH_1^{-1}H_2^{-1}\cdots H_k^{-1}$. Not only is A_k preconditioned but the Hermite basis of the first k columns of A_k have now been factored out. The purpose of this refinement is to ensure that the Hermite form of the principal $k\times k$ submatrix of A_k is generic, that is, has all diagonal entries 1 except for possibly the last. At stage k we now have

$$T_k = \left[\begin{array}{cc|c||c} I_{k-1} & h & \bar{h} & \cdots \\ & d & \bar{d} & \cdots \\ \hline & a & \bar{a} & \cdots \\ & b_1 & \bar{b}_1 & \cdots \\ & \vdots & \vdots & \cdots \\ & b_* & \bar{b}_* & \cdots \end{array}\right] \in \mathbb{Z}^{n\times m}, \tag{3.1}$$

Because $H_1,\ldots,H_k$ have been factored out of A_k, the first $k-1$ diagonal entries of T_k are 1, and also $h_k = \gcd(d,a,b_1,\ldots,b_*) = 1$. Like before, at the start of stage k only the first k columns of T_k and known, and the column

$$\left[\begin{array}{c} \bar{h} \\ \bar{d} \\ \hline \bar{a} \\ \bar{b}_1 \\ \vdots \\ \bar{b}_* \end{array}\right] \in \mathbb{Z}^{n\times(k+1)}$$

needs to be computed. Once this column is computed, we simply compute H_{k+1} as the Hermite basis of the first $k+1$ columns of T_k.

The refined algorithm is summarized as follows. First recover H_1 by computing the gcd of the entries in the first column of A. Like before, assume withot loss of generality the $A_{1\,1}$ is positive. Initialize $A_1 = AH_1^{-1}$ and proceed for $k = 1, 2, \ldots, m$ as follows:

1. Compute column $k+1$ of T_k.
2. Recover $H_k \in \mathbb{Z}^{m\times m}$ from the Hermite basis of the first $k+1$ columns of T_k.
3. Update $T_k \leftarrow H_{k+1}^{-1}$.
4. Compute C_k and Q_k from T_k.
5. Let the first $k+1$ columns of T_{k+1} be those of $Q_k C_k T_k$.
6. Set $A_{k+1} := C_k A_k H_{k+1}^{-1}$.

After completion $H_1, H_2, \ldots, H_m$ are recovered, which gives the Hermite form of A.

In terms of complexity, the dominant step in the algorithm is step 1, the computation of column $k+1$ of T_k. To simplify the discussion let us assume $n = m$. Then a single stage of steps 2–5 are accomplished with $O(n)$ operations on integers of bitlength bounded by $O(n(\log n + \log ||A_k||))$. A subtlety in step 6 is that the bitlength of entries in $A_1, A_2, A_3, \ldots$ could (in theory) grow large, although this is not observed in practice. Indeed, for many input matrices many of the H_* will be the identity matrix, and even if all H_* are nontrivial, experiments show that $\log ||A_k||$ grows only slightly compared to $\log ||A||$. Under the reasonable heuristic assumption that $\log ||A_k|| \in O(\log n + \log ||A||)$, the cost of step 1 is thus $O(n^3(\log n + \log ||A||)^2)$ bit operations, compared to $O(n)$ operations on integers bounded in bilength by $O(n(\log n + \log ||A||))$ for the remaining steps. Assuming pseudo-linear integer arithmetic, step 1 thus dominates the cost of the algorithm by a factor of n.

Let us examine step 1 in more detail, in particular the computation of

$$\begin{bmatrix} \bar{h} \\ \bar{d} \end{bmatrix} \in \mathbb{Z}^{k\times 1}.$$

At the start of stage k, write the preconditioned matrix A_k using a block decomposition as

$$A_k = \left[\begin{array}{c|c||c} B & b & \cdots \\ \hline F & f & \cdots \end{array}\right] \in \mathbb{Z}^{n\times(k+1)}, \tag{3.2}$$

where B is $k \times k$ and nonsingular. Then we have

$$\begin{bmatrix} \bar{h} \\ \bar{d} \end{bmatrix} = (1/d) \begin{bmatrix} I_{k-1} & h \\ & d \end{bmatrix} B^{-1}(db). \tag{3.3}$$

The main cost here is the computation of $B^{-1}(db)$. Using p-adic lifting this can be accomplished in $O(k^3(\log k + \log ||B||)^2)$ bit opearations. The next chapter defines a data structure that, if known, can in many cases allow me to compute $B^{-1}(db)$ in a running time that is factor of k faster. This faster linear solving algorithm described in the next chapter requires the Hermite form of B to be generic.

3.2 Utilizing Lattice Compression

Consider the matrix T_k shown in (3.1). Our next refinement is to show how to avoid computing most of the b_* and $\bar{b}_*$ entries. Consider the decomposition of matrix A_k shown in (3.2). Further decompose F as

$$F = \left[\begin{array}{c|c} \bar{F} & \bar{f} \end{array} \right] \in \mathbb{Z}^{(n-k)\times k}$$

where $\bar{f} \in \mathbb{Z}^{(n-k)\times 1}$ is the last column. Stage k now proceeds as follows. First we find $\bar{h}$ and $\bar{d}$, and then form the matrix

$$\left[\begin{array}{c|c|c||c} I_{k-1} & h & \bar{h} & \cdots \\ \hline & d & \bar{d} & \cdots \\ \hline \bar{F} & \bar{f} & f & \cdots \end{array} \right] \in \mathbb{Z}^{n\times m}. \tag{3.4}$$

which is left equivalent to T_k. To transform this matrix to have the form of (3.1) we need to zero out the block $\bar{F}$ to get the entries below d and $\bar{d}$ in columns k and $k+1$. In particular, we have

$$\begin{bmatrix} a & \bar{a} \\ b_1 & \bar{b}_1 \\ \vdots & \vdots \\ b_{n-k-1} & \bar{b}_{n-k-1} \end{bmatrix} = \left[\begin{array}{cc} \bar{f} & f \end{array} \right] - \bar{F} \left[\begin{array}{cc} h & \bar{h} \end{array} \right].$$

Since the bitlength of each entry of h and $\bar{h}$ can be large, the matrix vector products $\bar{F}h$ and $\bar{F}\bar{h}$ can be expensive, especially considering that the row dimension of $\bar{F}$ is $n-k$, which will be large for large n Empirically, we have observed that for many input matrices, already for a very small ℓ, for example $\ell = 3$, that the Hermite basis of

$$\left[\begin{array}{c|c|c} I_{k-1} & h & \bar{h} \\ \hline & d & \bar{d} \\ \hline & a & \bar{a} \\ & b_1 & \bar{b}_1 \\ & \vdots & \vdots \\ & b_\ell & \bar{b}_\ell \end{array}\right] \mathbb{Z}^{(k+\ell+1)\times(k+1)} \tag{3.5}$$

is equal to the principal $(k+1)\times(k+1)$ submatrix of H_{k+1}. Hence, a useful heuristic is to compute the pairs $(a,\bar{a}),(b_1,\bar{b}_1),\ldots$ in succession. If the Hermite basis of (3.5) ever becomes I_{k+1} we can stop.

We can ensure that the Hermite basis of (3.5) has converged for $\ell \ll n-k-1$ with high probability by using the lattice compression technique of Chen and Storjohann [2005]. Choose an integer parameter $\ell > 0$. (How to choose ℓ will be discussed shortly.) For any matrix $R \in \mathbb{Z}^{(m+\ell)\times n}$, the matrix

$$\bar{A} = \left[\begin{array}{c} RA \\ \hline A \end{array}\right] \in \mathbb{Z}^{(n+m+\ell)\times m}$$

will have the same Hermite form as A. Chen and Storjohann [2005] show that if entries in R are chosen uniformly and randomly from $\{0,1,\ldots,\lambda-1\}$, where λ is a multiple of six and satisifies

$$\lambda \geq 8 \times (25n\ \log 2nm||A||)^{\frac{1}{\lceil \ell/3 \rceil}},$$

then the probability that the Hermite basis of the principal $k \times \ell$ submatrix of $\bar{A}$ will *not* equal the Hermite basis of the first k columns of A is at most

$$\left(\frac{9}{10}\right)\left(\frac{1}{2}\right)^{\ell-1}.$$

The idea is now to contruct the matrix $\bar{A}$ as described above and run the algorithm on it instead of A. At each stage we now go to the next stage using the much smaller matrix (3.5) instead of the first $k+1$ columns of T_k. Since we have m stages the overall probablity of failure (at any stage) can be bounded by $0 < \tau < 1$ by choosing

$$\ell > \log\left(\frac{9m}{10\tau}\right) + 1.$$

For example, if $n = m = 10000$ and $||A|| = 99$, then to achieve an overall probability of success of at least $1/2$ we can choose $\ell = 11$ and $\lambda = 1452$.

Chapter 4

The Specialized Outer Product Adjoint with Applications

Let $A \in \mathbb{Z}^{n\times n}$ be nonsingular with generic Hermite form H, that is, all diagonals of H are one except for possibly the last. Then there exists a unique unimodular matrix $U \in \mathbb{Z}^{n\times n}$ such that

$$UA = H = \left[\begin{array}{c|c} I_{n-1} & h \\ \hline & d \end{array}\right] \in \mathbb{Z}^{n\times n}, \tag{4.1}$$

where $h \in \mathbb{Z}^{(n-1)\times 1}$ and $d = |\det A| \in \mathbb{Z}$ is the absolute value of the determinant of A. By Cramer's rule, the matrix dA^{-1} will be integral. For example, consider the input matrix

$$A = \begin{bmatrix} 38 & 63 & -12 & -21 & 82 \\ 91 & -26 & 45 & 90 & -70 \\ -1 & 30 & -14 & 80 & 41 \\ 63 & 10 & 60 & 19 & 91 \\ -23 & 22 & -35 & 88 & 29 \end{bmatrix} \in \mathbb{Z}^{5\times 5} \tag{4.2}$$

with determinant $d = 888309873$ and Hermite form

$$H = \begin{bmatrix} 1 & & & & 118556465 \\ & 1 & & & 237549876 \\ & & 1 & & 649715522 \\ & & & 1 & 48308716 \\ & & & & 888309873 \end{bmatrix} \in \mathbb{Z}^{5\times 5}.$$

Then

$$dA^{-1} = \begin{bmatrix} 12806982 & 9064115 & -46174901 & 5196584 & 34641287 \\ -845433 & -3779058 & 85932579 & -18399891 & -70484628 \\ -15764220 & -6453838 & 78729043 & -4646392 & -67730059 \\ -3485052 & 1369978 & 7994951 & 432715 & 500092 \\ 2348172 & -1890637 & -31054436 & 11159186 & 28315901 \end{bmatrix}.$$

While the total size (number of bits to represent) A is $O(n^2 \log ||A||)$, each entry of dA^{-1} will be (up to sign) a minor of A of dimension $n-1$, and thus by Hadamard's bound will have bitlength bounded by $O(n(\log n + \log ||A||))$, or about n times the bitlength of entries in A. The total size of dA^{-1} is thus $O(n^3(\log n + \log ||A||))$, or about n times the space required for A. Instead of representing dA^{-1} as a dense $n \times n$ matrix, Storjohann [2010] shows that dA^{-1} can be expressed as the outer product of a column and row vector. The outer product only captures $dA^{-1} \bmod d$, but if d is about the same bitlength as entries in dA^{-1}, then a large part of dA^{-1} will be known. We derive the construction now.

Considering the shape of H in (4.1), the unimodular matrix

$$V := \left[\begin{array}{c|c} I_{n-1} & -h \\ \hline & 1 \end{array}\right]$$

can be used to diagonalize H, that is,

$$UAV = \left[\begin{array}{c|c} I_{n-1} & \\ \hline & d \end{array}\right]. \tag{4.3}$$

Inverting both sides of (4.3), multiplying by d, and solving for dA^{-1} yields

$$dA^{-1} = V \left[\begin{array}{c|c} dI_{n-1} & \\ \hline & 1 \end{array}\right] U. \tag{4.4}$$

Consider taking (4.4) modulo d. The dI_{n-1} submatrix vanishes and we are left with

$$dA^{-1} = vu \bmod d, \tag{4.5}$$

where $v \in \mathbb{Z}^{n \times 1}$ is the last column of V and $u \in \mathbb{Z}^{1 \times n}$ is the last row of U, which is necessarily the last row of dA^{-1}.

We call the tuple $(v, u, d) \in (\mathbb{Z}^{n\times 1}, \mathbb{Z}^{1\times n}, \mathbb{Z}_{>0})$ the *specialized outer product adjoint (sopa)* of A. Note that the total size of the sopa is only $O(n(\log n + \log ||A||))$ bits, or about the same as required to write down the input matrix A.

Next we give a concrete example of a sopa.

Example 2. *The sopa for the example matrix A in (4.2) is given by d together with the row vector*

$$u = \left[\begin{array}{ccccc} 2348172 & -1890637 & -31054436 & 11159186 & 28315901 \end{array} \right]$$

and column vector

$$v = \left[\begin{array}{c} -118556465 \\ -237549876 \\ -649715522 \\ -48308716 \\ 1 \end{array} \right].$$

Indeed, we have

$$vu \ mod \ d = \left[\begin{array}{rrrrr} 12806982 & 9064115 & -46174901 & 5196584 & 34641287 \\ -845433 & -3779058 & 85932579 & -18399891 & -70484628 \\ -15764220 & -6453838 & 78729043 & -4646392 & -67730059 \\ -3485052 & 1369978 & 7994951 & 432715 & 500092 \\ 2348172 & -1890637 & -31054436 & 11159186 & 28315901 \end{array} \right]$$

where the mod operation reduced integers in the symmetric range modulo d. For this example $vu \bmod d$ is actually equal to dA^{-1}, but this might not always be the case. In general, though, we always have

$$dA^{-1} = (vu \bmod d) + Ed$$

for an integer matrix E. (In this example E is the zero matrix.) If A is well conditioned then E will have very small entries and can be computed efficiently using using p-adic lifting or Chinese remaindering. However, our motivation here is not to compute the adjoint of A but to apply the sopa to solve linear systems.

The rest of this chapter is organized as follows. In Section 4.1 we show how to use the sopa for A to solve a nonsingular system $Ax = b$ for x in nearly optimal time, provided that A is well conditioned. Our algorithm in the subsequent chapter will requires the sopa for the principal $k \times k$ submatrices of an input matrix for $k = 1, 2, 3, \ldots$ in succession. In Section 4.2 we show how to compute the sopa for stage $k + 1$ given the sopa for stage k.

4.1 Nonsingular Linear System Solving

The nonsingular linear system solving problem takes as input a nonsingular matrix $A \in \mathbb{Z}^{n\times n}$, together with a column vector $b \in \mathbb{Z}^{n\times 1}$, and asks for the solution vector $A^{-1}b \in$

$\mathbb{Q}^{n\times 1}$. If an associate d of $\det A$ is known, the problem can be simplified somewhat by solving for $A^{-1}(db)$, which will be integeral. Actually, we will prefer to rewrite

$$A^{-1}(db) = (dA^{-1})b$$

since we are going to assume here that A has a generic Hermite form, and that we have on hand the sopa $(v, u, d) \in (\mathbb{Z}^{n\times 1}, \mathbb{Z}^{1\times n}, \mathbb{Z}_{>0})$ for A. Exploiting the fact that $dA^{-1} \equiv vu \bmod d$, computing $(dA^{-1})b$ now has three steps:

1. Compute

$$s := vub \bmod d = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \begin{bmatrix} u_1 & u_2 & \cdots & u_n \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \bmod d$$

   with entries in s reduced modulo d in the symmetric range. Obviously, the dot product of u and b should be computed first!

2. Compute $r := b - As(1/d) \in \mathbb{Z}^{n\times 1}$.

3. Compute $e := A^{-1}r \in \mathbb{Z}^{n\times 1}$ using some other method such as p-adic lifting or Chinese remaindering.

We claim that the solution vector is then $(dA^{-1})b = s + ed$. To understand the construction consider step 2. Since $(dA^{-1}) \equiv vu \bmod d$ we know that $(dA^{-1})b = s + ed$ for some integer vector $e \in \mathbb{Z}^{n\times 1}$. Solving for e yields the formula for the residue r. It turns out that $||r||$ will always be small. In particular, since s has entries reduced modulo d, we have $||s(1/d)|| < 1$, and

$$\begin{aligned} ||r|| &= ||b - As(1/d)|| \\ &\leq ||b|| + ||A||_\infty, \end{aligned} \tag{4.6}$$

where $||A||_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}|$. Thus, r can be computed modulo an integer q that is relatively prime to d and satisfies $q \geq 2(||b|| + ||A||_\infty) + 1$. In step 3, since a large part of the solution vector has been recovered in step 1, e is expected to have small entries. The method is best illustrated with an example.

Example 3. *Let $A \in \mathbb{Z}^{5\times 5}$ be the example matrix from (4.2). Let*

$$b := \begin{bmatrix} -4 \\ 5 \\ -91 \\ -44 \\ -38 \end{bmatrix} \in \mathbb{Z}^{5\times 1}.$$

Step 1 computes

$$\begin{aligned} s &= vub \bmod d \\ &= \begin{bmatrix} -118556465 \\ -237549876 \\ -649715522 \\ -48308716 \\ 1 \end{bmatrix} \begin{bmatrix} 2348172 & -1890637 & -31054436 & 11159186 & 28315901 \end{bmatrix} \begin{bmatrix} -4 \\ 5 \\ -91 \\ -44 \\ -38 \end{bmatrix} \bmod d \\ &= \begin{bmatrix} -118556465 \\ -237549876 \\ -649715522 \\ -48308716 \\ 1 \end{bmatrix} 351789508 \bmod 888309873 \\ &= \begin{bmatrix} -13939583 \\ 94182186 \\ 86177632 \\ 143516474 \\ 351789508 \end{bmatrix} \bmod 888309873 \end{aligned}$$

Step 1 thus consists of producing the scalar equal to the dot product of u and b, and then multiplying the vector v by this scalar, working modulo d throughout. Assuming pseudo-linear intreger arithemtic, the cost of step 1 is thus nearly optimal, that is, within a polylogarithmic factor of the space required to represent an n-dimensional vector filled with integers reduced modulo d.

To perform step 2 efficiently we choose a modulus q that is relatively prime to d and large enough to capture entries in r in the symmetric range modulo q. In this example we can choose $q = 1009$. Then we first reduce entries in $s(1/d)$ modulo q before performing the matrix vector product:

$$
\begin{aligned}
r &= b - As(1/d) \\
&= \begin{bmatrix} -4 \\ 5 \\ -91 \\ -44 \\ -38 \end{bmatrix} - \begin{bmatrix} 38 & 63 & -12 & -21 & 82 \\ 91 & -26 & 45 & 90 & -70 \\ -1 & 30 & -14 & 80 & 41 \\ 63 & 10 & 60 & 19 & 91 \\ -23 & 22 & -35 & 88 & 29 \end{bmatrix} \begin{bmatrix} -13939583 \\ 94182186 \\ 86177632 \\ 143516474 \\ 351789508 \end{bmatrix} \frac{1}{888309873} \\
&\equiv \begin{bmatrix} -4 \\ 5 \\ -91 \\ -44 \\ -38 \end{bmatrix} - \begin{bmatrix} 38 & 63 & -12 & -21 & 82 \\ 91 & -26 & 45 & 90 & -70 \\ -1 & 30 & -14 & 80 & 41 \\ 63 & 10 & 60 & 19 & 91 \\ -23 & 22 & -35 & 88 & 29 \end{bmatrix} \begin{bmatrix} 361 \\ 152 \\ 230 \\ 231 \\ 166 \end{bmatrix} \bmod 1009 \\
&\equiv \begin{bmatrix} -38 \\ 18 \\ -122 \\ -89 \\ -63 \end{bmatrix} \bmod 1009
\end{aligned}
$$

Finally, step 3 uses either Chinese remaindering or p-adic lifting to compute the solution e to $Ae = r$. We obtain

$$
e = \begin{bmatrix} 3 \\ -5 \\ -5 \\ -1 \\ 1 \end{bmatrix}.
$$

The system solution $(dA^{-1})b$ is then obtained as

$$(dA^{-1})b = s + ed = \begin{bmatrix} -13939583 \\ 94182186 \\ 86177632 \\ 143516474 \\ 351789508 \end{bmatrix} + \begin{bmatrix} 3 \\ -5 \\ -5 \\ -1 \\ 1 \end{bmatrix} 888309873 = \begin{bmatrix} 2650990036 \\ -4347367179 \\ -4355371733 \\ -744793399 \\ 1240099381 \end{bmatrix}.$$

4.2 Updating the SOPA

Suppose that we have the sopa $(v, u, d) \in (\mathbb{Z}^{n\times 1}, \mathbb{Z}^{1\times n}, \mathbb{Z}_{>0})$ for an $A \in \mathbb{Z}^{n\times n}$ that has a generic Hermite form. Suppose further that A is the principal $n \times n$ submatrix of

$$\bar{A} := \left[\begin{array}{c|c} A & b \\ \hline c & a \end{array}\right] \in \mathbb{Z}^{(n+1)\times(n+1)}.$$

Under the assumption that $\bar{A}$ also has a generic Hermite form, we show how to compute the sopa for $\bar{A}$.

Recall that, by definition, the components $v \in \mathbb{Z}^{n\times 1}$ and $d \in \mathbb{Z}_{>0}$ of the sopa of A define the Hermite form of A, and vice versa. In particular, if

$$v = \left[\begin{array}{c} -h \\ \hline 1 \end{array}\right] \in \mathbb{Z}^{n\times 1}$$

then the Hermite basis of A is

$$H = \left[\begin{array}{c|c} I_{n-1} & h \\ \hline & d \end{array}\right].$$

We also know that $u \in \mathbb{Z}^{1\times n}$ is the last row of dA^{-1}. These observations apply also to the sopa of $\bar{A}$. Thus, computing the sopa of $\bar{A}$ boils down to the following computations.

1. Compute the Hermite form $\bar{H}$ of $\bar{A}$.

2. Compute the last row of $e\bar{A}^{-1}$ where $e = |\det \bar{A}|$, the last diagonal entry of $\bar{H}$.

For step 1 we first apply to the first n rows of $\bar{A}$ the unimodular matrix which transforms A to Hermite form. This gives the left equivalent matrix

$$\left[\begin{array}{c|c} H & (1/d)H(dA^{-1})b \\ \hline c & a \end{array}\right] \in \mathbb{Z}^{(n+1)\times(n+1)}$$

which has the shape

$$\left[\begin{array}{c|c|c} I_{n-1} & h & * \\ \hline & d & * \\ \hline *_1 & *_2 & * \end{array}\right] \in \mathbb{Z}^{(n+1)\times(n+1)}. \tag{4.7}$$

where

$$c = \left[\begin{array}{c|c} *_1 & *_2 \end{array}\right].$$

The main cost of computing (4.7) is to compute $(dA^{-1})b$; this is done using the sopa of A as described in the previous section. Matrix (4.7) can now be transformed to Hermite form by using I_{n-1} to zero out $*_1$, and then performing some additional operations on only the last two columns. The cost of transforming (4.7) to Hermite form is thus $O(n)$ operations on integers bounded in length by $O(n(\log n + \log ||\bar{A}||))$ bits.

For step 2, note that the last row of $e\bar{A}^{-1}$ is given by

$$\left[\begin{array}{c|c} -c(dA^{-1}) & d \end{array}\right] \in \mathbb{Z}^{1\times(n+1)}.$$

The computation of $c(dA^{-1})$ can also make use of the sopa of A.

Chapter 5

Conclusions

We begin by giving an overview of the complete Hermite form algorithm.

Let $A \in \mathbb{Z}^{n \times m}$ be a full column rank input matrix. Assume that we have already used the lattice compression technique described in Section 3.2 so that, with high probability, the Hermite basis of the principal $(k+\ell) \times k$ matrix of A is the Hermite basis of the first k columns of A, for $k = 1, 2, \ldots, m$.

At the start of stage k we have the preconditioned input

$$A_k = C_{k-1} \cdots C_2 C_1 A H_1^{-1} H_2^{-1} \cdots H_k^{-1}$$

which can be written as a block decomposition as

$$A_k = \left[\begin{array}{c|c||c} B & b & \cdots \\ \hline F & f & \cdots \end{array}\right] \in \mathbb{Z}^{n \times m}$$

were B is $k \times k$ and $b \in \mathbb{Z}^{k \times 1}$. We also have the Hermite basis of B which has the shape

$$\left[\begin{array}{c|c} I_{k-1} & h \\ \hline & d \end{array}\right] \in \mathbb{Z}^{k \times k}.$$

Finally, we also have the sopa for B.

To get to stage $k+1$ we proceed as follows. First, use the sopa based linear solving algorithm described in Section 4.1 to compute the last column of the matrix

$$\left[\begin{array}{c|c|c} I_{k-1} & h & \bar{h} \\ \hline & d & \bar{d} \end{array}\right] \in \mathbb{Z}^{k \times (k+1)}$$

which is left equivalent to

$$\left[\begin{array}{c|c} B & b \end{array}\right] \in \mathbb{Z}^{k\times(k+1)}.$$

If we decompose

$$F = \left[\begin{array}{c|c} \bar{F} & \bar{f} \end{array}\right]$$

where $\bar{f} \in \mathbb{Z}^{k\times 1}$, then we know the matrix

$$\left[\begin{array}{c|c|c||c} I_{k-1} & h & \bar{h} & \cdots \\ \hline & d & \bar{d} & \cdots \\ \hline \bar{F} & \bar{f} & f & \cdots \end{array}\right] \in \mathbb{Z}^{n\times m}$$

is left equivalent to A_k. Use I_{k-1} to zero out the first ℓ rows of $\bar{F}$ to obtain the left equivalent matrix

$$\left[\begin{array}{c|c|c||c} I_{k-1} & h & \bar{h} & \cdots \\ \hline & d & \bar{d} & \cdots \\ \hline & a & \bar{a} & \cdots \\ & b_1 & \bar{b}_1 & \cdots \\ & \vdots & \vdots & \cdots \\ & b_\ell & \bar{b}_\ell & \cdots \\ \hline * & * & * & \cdots \end{array}\right] \in \mathbb{Z}^{n\times m}. \tag{5.1}$$

The remainder of the computation from stage k to stage $k+1$ can proceed using only the principal $(k+\ell+1)\times(k+1)$ submatrices of A_k and the matrix in (5.1). A failure of the lattice conditioning is detected by noticing that $A_{k+1} = C_k A_k H_{k+1}^{-1}$ is not integral.

We have implemented the algorithm just described in in Maple. We offer here some timings compared to Maple's implementation of Hermite form, where $-$ means there is no return in limited time.

And also, we observe that

- when n is doubled from 250 to 500, the running time grows by about 4.77 times.
- when n is doubled from 500 to 1000, the running time grows by about 7.25 times;
- when n is doubled from 1000 to 2000, the running time grows by about 9.70 times.

Empirically, the running time is proportional to $O(nm^2)$ operations on integers bounded in bitlength by $O(n(\log n + \log ||A||))$ bits.

n	New	Maple
100	0.973	1.619
250	3.315	25.727
500	15.818	586.924
1000	114.689	–
2000	1111.973	–
5000	26569.598	–

Table 5.1: Time in seconds for new algorithm and `LinearAlgebra[Hermite]` in Maple 16

References

Z. Chen and A. Storjohann. A BLAS based C library for exact linear algebra on integer matrices. In M. Kauers, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'05*, pages 92–99. ACM Press, New York, 2005.

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.

P. D. Domich. *Residual Methods for Computing Hermite and Smith Normal Forms*. PhD thesis, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 1985.

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.

Ming S. Hung. An application of the hermite normal form in integer programming. *Linear Algebra and its Applications*, page 163179, 1990. doi: https://doi.org/10.1016/0024-3795(90)90228-5.

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.

Daniele Micciancio. Improving lattice based cryptosystems using the hermite normal form. *Cryptography and Lattices Conference 2001*, 2001. doi: 10.1007/3-540-44670-2_11.

C. Pauderis and A. Storjohann. Deterministic unimodularity certification. In J. van der Hoeven and M. van Hoeij, editors, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'12*, pages 281–288. ACM Press, New York, 2012.

A. Storjohann. A fast, practical and deterministic algorithm for triangularizing integer matrices. Technical Report 255, Departement Informatik, ETH Zürich, December 1996.

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

A. Storjohann. The modulo extended gcd problem and space efficient algorithms for integer matrix computations, 2003. Submitted.

A. Storjohann. On the complexity of inverting integer and polynomial matrices. *Computational Complexity*, 2010. Accepted for publication.

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.