Hermite Normal Forms and its Cryptographic Applications

A thesis submitted in fulfillment of the
requirements for the award of the degree

Master of Computer Science

from

UNIVERSITY OF WOLLONGONG

by

Vasilios Evangelos Tourloupis

School of Computer Science and Software Engineering
May 2013

Dedicated to

My Family

Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

Vasilios Evangelos Tourloupis
May 23, 2013

Abstract

Hermite Normal Form (HNF) matrices are a standard form of integer matrices used in applications such as lattice based cryptography and integer programming. Calculating the HNFs of randomly selected matrices, however, is not efficient and, although polynomial algorithms exist to address this issue, they do not address the private and public key selection process that attempts to search for a reduced public key size. At best, searching for a matrix (private key) whose HNF (public key) is in 'minimal' form (that is, it can be expressed as a single column of values) is a trial and error process with only a 40% chance of success.

In this thesis, a way of reducing the trial and error associated with this process is studied. It does so by taking advantage of the incremental nature of Micciancio and Warinschi's algorithm for calculating HNF matrices row by column. The first contribution of this thesis reveals a stochastic observation relating to prime determinants. Specifically, if a leading principal minor of a matrix is prime (that is, if the determinant of a $k \times k$ sub-matrix of an $n \times n$ matrix is prime), it increases the probability of the next (or $(k+1)$-th) leading principal minor also being prime. Test results have revealed a reduction in the average number of steps of selecting a matrix with a prime determinant by a factor of 12 when compared to the traditional trial and error techniques of selecting such a matrix.

Since the determinant of a matrix does not need to be prime in order for its HNF to be in 'minimal' form, an alternative way of reducing the public key matrix is also studied, again taking advantage of the incremental nature of Micciancio and Warinschi's algorithm for calculating HNF matrices.

Finally, this thesis also looks at an alternative way of selecting private key matrices whose HNF public key matrices are in 'minimal' form. It does so by using a simple sieving method, which filters prime (or probable prime) determinants of a matrix. With a success rate of about 99%, this method leads more directly to a solution rather than a subset of possible solutions.

Acknowledgement

I wish to thank Dr. Thomas Plantard for the technical support in helping me to identify and establish the mathematical phenomenon that forms the basis of this thesis, as well as for the valuable feedback provided in writing it. I also wish to thank Professor Willy Susilo for his supervision and support in overseeing this research, and for his valuable feedback. I would also like to thank Dr. Christopher Doche for his suggestion concerning the efficiency of the main algorithm. I further like to thank my parents for their financial and moral support in helping me to complete this project. Finally, I would like to thank fellow co-students and staff at the University of Wollongong for their advice and support during this project.

Publications

V. Tourloupis, T. Plantard, and W. Susilo, "Computing Hermite Normal Forms for Cryptography." Submitted to Theoretical Computer Science.

Other Publications

V. Tourloupis and L. Wang, "On the state of modern hash functions," *TELECOM MARKET*, vol. 11, November 2009.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Lattice based cryptography is an important area of study in modern cryptography. The motivation for introducing lattice-based cryptosystems is twofold. First, it is certainly of interest to have cryptosystems based on a variety of hard mathematical problems, since a breakthrough in solving one mathematical problem does not compromise the security of all systems. Second, lattice-based cryptosystems are frequently much faster than factorisation or discrete logarithm-based systems like ElGamal, RSA and ECC. Furthermore, the simple linear algebra operations used by lattice-based systems are very easy to implement in hardware and software.

1.1 Lattice Based Cryptography

In Crypto '97, Goldreich, Goldwasser and Halevi proposed a one-way trapdoor function called GGH from which they could derive public-key encryption and digital signatures. The security of this scheme was based on the conjectured computational difficulty of lattice problems, thereby providing a possible alternative to existing public-key encryption algorithms and digital signature schemes. In particular, the scheme relied on the Closest Vector Problem (CVP) as its underlying hard problem to construct trapdoor functions.

The methods described by Goldreich *et al.* were asymptotically more efficient than the RSA and ElGamal encryption schemes, for example, in that the computation time for encryption, decryption, signing, and verifying were all quadratic in the natural security parameter [GGH97]. Specifically, for the security parameter of k, the new scheme required a computational time of $O(k^2)$ to execute, compared to the RSA and ElGamal schemes, which required a computational time of $O(k^3)$[1].

[1]Note that more advanced techniques provided better results. For instance, RSA encryption/decryption requires $O(kM(k))$ where $M(k)$ is the complexity of multiplying two values modulo

The size of the public key, however, was longer: $O(k^2)$ compared to $O(k)$ for the RSA and ElGamal schemes. The authors of this new scheme, however, argued that given today's technologies, the increase in key size is more than compensated by the decrease in computational time.

The GGH signature scheme, for example, can be described in three steps:

> Setup: Compute a 'good basis', G, and a 'bad basis', B, of a lattice $\mathcal{L}$, such that both span the same lattice $\mathcal{L}$, (i.e. $\mathcal{L}(G) = \mathcal{L}(B)$). Provide B as a public basis (key) and keep G as a private basis (key).
>
> Sign: Use the good basis to calculate an efficient approximation of the closest vector to a vector. In this case, the initial vector is the *message* and the approximation is the *signature*.
>
> Verify: Check if the approximation is in the lattice spanned by the bad basis.

Note that GGH uses the first of Babai's methods [Bab86] to approximate CVP. That is $s = \lceil mG^{-1} \rfloor G$, where $\lceil x \rfloor$ represents the closest integer of x if x is real, and the vector $(\lceil x_0 \rfloor, \lceil x_1 \rfloor, ..., \lceil x_{n-1} \rfloor)$ if x is a vector of $\mathbb{R}^n$. The important points for the security and efficiency of this cryptosystem to consider as follows.

i) It is easy to compute a 'bad basis' from a "good basis", but difficult to compute a 'good basis' from a 'bad basis'.

ii) It is easy to compute a good approximation of CVP with a 'good basis', but difficult to do so with a 'bad basis'.

iii) It is easy to check the inclusion of a vector in a lattice even with a 'bad basis'.

In 1999, Nguyen demonstrated a major flaw in the design of this scheme which had two implications: any ciphertext leaked information on the plaintext, and the problem of decrypting ciphertexts could be reduced to a special closest vector problem which was much easier to solve than the general problem. In particular, the given vector was very close to the lattice, which made it possible in practice to find the closest vector by standard techniques [Ngu99]. Due to this attack, the utilization of GGH required that a lattice be defined with dimension greater than 500 to

---

N of size k bits. Also, with Montgomery representation (allowing for a fast reduction) and Fast Fourier Transforms, $M(k) = O(k \log k \log \log k)$.

ensure its security. Nguyen suggested other solutions that would prevent the flaw but concluded that, even modified, the scheme could not provide sufficient security without being impractical.

1.2 HNF to Improve Lattice Based Cryptography

In 2001, Micciancio [Mic01] proposed some major improvements to the speed and security of GGH. In particular, he showed how to build lattice based trapdoor functions with public key sizes of 330KB and less, and still be secure against the best known lattice attacks. He did so by using Hermite Normal Form (HNF) matrices, which are upper triangular matrices analogous to reduced row echelon form matrices over $\mathbb{Z}$. Specifically, a square matrix, $H = (h_{i,j})$, with integer entries is in Hermite Normal form if it satisfies the following conditions[2].

1. H is an upper triangular matrix, i.e. $h_{i,j} = 0$ if $i > j$,
2. For every i, $h_{i,i} > 0$ (i.e. its diagonal entries, $h_{i,i}$, are positive),
3. For every $i > j$, we have $0 \leq h_{i,j} < h_{i,i}$ (i.e. in a given column, the entries above the diagonal are less than the diagonal coefficient, and at least zero).

Also, for every integer matrix A, there exists a unimodular matrix, U, such that $H = A \cdot U$, and H is unique [Coh93].

The following matrix is in Hermite Normal Form.

$$H = \begin{bmatrix} 2 & 0 & 1 & 11 \\ 0 & 1 & 5 & 12 \\ 0 & 0 & 6 & 12 \\ 0 & 0 & 0 & 33 \end{bmatrix}$$

Notice that all off diagonal entries in the lower half of the matrix are zero (condition 1); all diagonal entries are greater than zero (condition 2); and all off diagonal entries in the upper half of the matrix are greater than or equal to zero, but less than the value of the entry on the diagonal (condition 3).

Note that, some authors (like Micciancio and Warinschi) use lower triangular matrices, rather than upper triangular matrices, in which case suitable adjustments

[2]This is different from the traditional definition of HNF, however, there is an easy transformation to go from one to the other. See [Coh93] for further details.

need to be made to the rest of the definition. This will depend on how the matrices are represented. Typically, matrices that are represented as column vectors have their HNF represented as a lower triangular matrix, while matrices that are represented as row vectors have their HNF being represented as an upper triangular matrix (as above) [Coh93].

The advantage of using Hermite Normal Form matrices is that they improve the efficiency of cryptosystems in terms of key length and computation time without compromising their security [PSW08]. And, if used for digital signature schemes, they are better for verifying digests, as well as making it more difficult to recover the private key [Mic01] [MW01] [PSW08].

In 2003, NTRUSign [HHGP$^+$03] was developed based on similar methods to GGH, but with improvements to the CVP approximation algorithm. In particular, Hoffstein *et al.* noted that such approximation algorithms do not provide a zero-knowledge scheme since the transcript of signatures leaks information about the private key. They introduced a *perturbation technique* as a general way of reducing the effectiveness of transcript analysis in CVP-based signature schemes. In the case of NTRUSign, the use of perturbations guaranteed that the number of signatures required to extract useful information far exceeded practical requirements.

They also proposed a slightly different class of convolution modular lattices, referred to as *transpose NTRU lattices*, that seemed to be more resistant to previously known attacks. Nevertheless, in 2006, Nguyen and Regev proposed a general attack against both the GGH signature scheme and NTRUSign [NR06]. This clever attack used large CVP approximations to recover the parallelepiped that made up the private key.

In 2008, Plantard *et al.* proposed a digital signature scheme similar to that of GGH, except that it uses the l_∞-norm to construct digital signatures, instead of the l_2-norm used by GGH. The advantage of this scheme is that it increases the security of the resulting digest, making it difficult to recover the parallelepiped that makes up the private key [PSW08]. It, too, uses HNF matrices to reduce the public key. Like GGH, it also provides for a more efficient implementation.

1.3 HNF Computations

As a comparison, the simplest algorithms for computing the Hermite Normal Form use a procedure similar to the Euclidean algorithm by replacing the diagonal element

in a given row by the gcd of the elements in the same row. This approach requires $O(m^2n)$ operations on rational numbers, where m and n are the number of rows and columns of the matrix, respectively. This method, however, does not properly bind the coefficients of intermediate results, even in very reasonable situations [Coh93]. For example, in [HM90], Hefner and McCurley provide a 20×20 integer matrix whose coefficients are less than or equal to 10, but needs integers of up to 1500 decimal digits to perform the calculation.

When $m \leq n$ and A is a matrix of rank m (in which case, H is an upper triangular matrix with non-zero determinant D), an important improvement suggested by several authors is to work modulo a multiple of the determinant (see [KB79] and [HM91] for examples). The first published algorithm to use this technique was by Frumkin [Fru77]. He based it on an algorithm for solving linear Diophantine equations, and had a running time that was bounded by a polynomial of $O(n^6)$ in the length of the input. Kannan and Bachem published another polynomial time algorithm shortly after [KB79]. Their method rearranged the order of operations of the classical Hermite Normal Form algorithm, and in doing so were able to prove a polynomial space and time bound for the algorithm. Chou and Collins gave a better space and time bound by modifying the Kannan-Bachem procedure [CC82], as did Iliopoulos who extended this procedure by using modular techniques [Ili89].

In the case of square matrices, Domich, Kannan and Trotter limit the size of the entries by modulo computation. They do so by first performing Gaussian elimination modulo the determinant, and then recovering the Hermite Normal Form of the original matrix by applying unimodular operations [DKT87]. Hafner and McCurley extend this result to non-square matrices, and also show how to use fast matrix multiplication in computing a triangular form of the input matrix [HM91]. In particular, they demonstrate that the running time of the algorithm is $O(m^3n(\log\log m)^3)$.

An even faster algorithm for computing the HNF of a triangular matrix is given by Storjohann in [Sto96]. The running time of the HNF algorithm obtained this way, which is dominated by the Hafner-McCurley triangularisation procedure, is equal to $O(n^{2+\theta}\log^2 M)$, where M is a bound on the entries of A, and n^θ is the number of arithmetic operations required to multiply two $n \times n$ matrices[3]. Although the space efficiency is not analysed, it follows from the triangularisation procedure that the space requirement of this, and all previous HNF algorithms, is $O(n^3 \log M)$ [MW01].

[3]Often referred to as the matrix exponent, Coppersmith and Winograd [CW87] were able to reduce θ to 2.376. In 2012, Williams [Wil12] reduced this to 2.3727.

			Time Complexity		Space Complexity	
Author	Year	Method	$m < n$	$m = n$	$m < n$	$m = n$
Frumkin	1977	Solving Linear Diophantine Equations	N/A	N/A		$O(n^6)$
Kannan-Buchem	1979	Classical HNF Algorithm	N/A	$O(2^{2n}n^{20n^3}\|B^1\|^{12n^3})$	N/A	$O^{\sim}(n^3\log(n))$
Chou-Collins	1982	Kannan-Buchem method	$O^{\sim}(n^5(m+n))$	$O^{\sim}(n^6)$	N/A	$O^{\sim}(n\log(n))$
Domich-Kannan-Trotter	1987	Modulo Computation	N/A	$O(n^3)$	N/A	$O^{\sim}(n\log(n))$
Iliopoulos	1989	Kannan-Buchem + Modular Techniques	N/A	$O(n^{5+\theta})$	N/A	$O^{\sim}(n\log(n))$
Hafner-McCurley	1989	Domich-Kannan-Trotter and Iliopoulos methods	$O(m^3n(\log\log m)^3)$	$O^{\sim}(n^4(\log\log(n))^3)$	N/A	$O^{\sim}(n\log(n))$
Storjohann	1996	Hafner-McCurley	N/A	$O^{\sim}(n^{2+\theta}\log^2(n))$	N/A	$O^{\sim}(n^3\log(n))$
Micciancio-Warinschi	2001	Hafner and McCurley method	$O^{\sim}(mn^4\log^2 n)$	$O^{\sim}(n^5\log^2(n))$	N/A	$O^{\sim}(n^2\log(n))$

Table 1.1: Time and space complexity of algorithms used to calculate Hermite Normal Form matrices measured in terms of the matrix dimension, n, by letting $M = O(n^c)$ for some constant c, and $m = O(n)$. Such conversions will be denoted by $O^{\sim}(*)$.

In [MW01], Micciancio and Warinschi were able to reduce the space bound to $O(n^2 \log M)$ (i.e. essentially the same size as the input matrix). They do so by using two space saving techniques. The first uses the Chinese Remainder Theorem on row vectors with appropriately selected prime numbers, while the second technique uses the Extended Euclidean Algorithm modulus the determinant to derive the column vectors. These space saving techniques also reduce the running time bound to $O(n^5 \log^2 M)$, although they also propose a heuristic version of their algorithm that further reduces the running time to $O(n^4 \log^2 M)$, or even $O(n^3 \log^2 M)$ based on whether or not certain conditions of the input matrix hold.

Steel [Ste] has reportedly been able to reduce the time complexity in Magma by extending the ideas of Micciancio and Warinschi and by using high performance linear algebra code within Magma [BCP97, BCFS10]. His techniques were further extended by Pernet and Stein [PS10], who have also reportedly been able to reduce the time complexity in Sage [S$^+$07] by improving on the heuristic version of Micciancio and Warinschi's implementation. They do so by defining a double determinant function which reduces the system of equations required for calculating the determinant of the sub-matrices that constitute the HNF matrix. Although they do provide timings that demonstrate that their implementation is asymptotically faster than other available versions (for example, compared to NTL [Sho09], Magma, GAP [S$^+$97] and Pari [Bat11]), it appears, at least for square matrices, that Magma is still superior in its performance for matrices whose elements are bound by eight bits. Furthermore, their implementation for square matrices does not appear to handle small matrix dimensions[4].

1.4 Motivation and Objectives

Although the techniques for calculating the HNF of a matrix have improved, the aim of finding a matrix whose HNF matrix can be expressed in 'optimal' form has not. That is, a HNF matrix whose triangular form is a single column of values. For

[4]This is a limitation of Micciancio and Warinschi's heuristic implementation of the HNF algorithm. The space efficient version of the algorithm overcomes this limitation. For further details, see [MW01]

instance, the following HNF matrix is in optimal form.

$$H = \begin{bmatrix} 1 & 0 & 0 & 335 \\ 0 & 1 & 0 & 286 \\ 0 & 0 & 1 & 1663 \\ 0 & 0 & 0 & 2873 \end{bmatrix}$$

This, for example, is useful for reducing the public key of a lattice based cryptosystem. According to Rose *et al.* [RPS11], the chances of finding such a matrix at random is approximately 40%. This means that, if the matrix were to be discarded every time its HNF was found not to be optimal, it would take between two to three times longer than the time for calculating its HNF, using Micciancio and Warinschi's heuristic technique.

The solution of Rose *et al.* was to calculate the determinant of the matrix prior to calculating its HNF, the assumption being that if the determinant were prime, its HNF would be in optimal form. This is because calculating the determinant is reduced to multiplying the diagonal entries of the matrix. This is because a prime determinant will have no factors, meaning that all coefficients on the diagonal except for the determinant will be equal to 1. In terms of solving linear diophantine equations, this means that if there are no trivial solutions, the determinant will be the last coefficient on the diagonal. This, therefore, reduces the complexity of finding an optimal HNF to that of calculating its determinant. That is, approximately two to three times the order of calculating the determinant, plus the time it takes to calculate its HNF. Although this does reduce the complexity of finding an optimal solution, there is still room to reduce a lot of the trial and error associated with the process.

1.5 Contributions

This thesis explores ways of reducing the trial and error associated with this process. It does so by utilising a property that encourages the determinant of a randomly chosen matrix to be prime. The idea is to ensure that the determinant of the matrix remains prime during its construction, and by doing so improves the likelihood of the target matrix determinant also being prime. This thesis also explores other ways of optimising HNF matrices, which does not involve prime determinants. The idea here is ensure that the HNF of randomly selected matrices remain optimal during their

construction, and by doing so improves the likelihood of the target HNF matrices also being optimal.

While the aim of this thesis is to optimise public key construction of the digital signature scheme proposed by Plantard *et. al.* [PSW08], it can also be adapted to the public key construction of other homomorphic cryptosystems proposed by Gentry *et al.* [GPV08] and Smart *et al.* [SV10]. It can further be adapted to the resolution of linear diophantine equations [CC82, Dwo98], as well as to loop optimisation techniques proposed by Ramanujam [Ram95].

1.6 Thesis Structure

This thesis is organised into five chapters, as illustrated in Table 1.2. The contents of each chapter are discussed below.

	Chapter 1	Introduction
Background	Chapter 2	Hermite Normal Forms
	Chapter 3	Distribution of Prime Determinants
Contributions	Chapter 4	Selecting an Optimal HNF
	Chapter 5	Conclusion and Future Work

Table 1.2: Structure of the thesis.

Chapter 1 provides readers with an introduction to this thesis, at the same time addressing its objectives and motivation. It also highlights its contributions, as well as its structure.

Chapter 2 provides a concise definition of an HNF matrix, as well as the different algorithms relating to its construction.

Chapter 3 looks at the asymptotic distribution of prime numbers, and how they relate to the distribution of prime determinants of a matrix.

Chapter 4 presents the contributions of this thesis; namely, a method of selecting an integer matrix such that its HNF is in optimal form. Typically, selecting a matrix whose determinant is prime will result in such an HNF matrix, however, this chapter also looks at other methods of selecting integer matrices which does not entail prime determinants.

Chapter 5 summarises the contributions of this thesis, and discusses direction for future work.

Chapter 2

Hermite Normal Forms

As noted in Chapter 1, the advantage of using Hermite Normal Form matrices is that they improve the efficiency of lattice based cryptosystems in terms of key length and computation time without compromising their security. This chapter provides a concise definition of an HNF matrix, as well as different algorithms relating to its construction.

2.1 Hermite Normal Form

A classical result of Hermite states that for every integer matrix A, there exists a unimodular matrix, U, such that $H = U \cdot A$ is an upper triangular matrix which is analogous to reduced row echelon form matrices over $\mathbb{Z}$, and is unique [Coh93]. Such matrices are known as Hermite Normal Forms, and are used in applications such as integer programming [HR90], loop optimisation [Ram95], and for finding the solution to a system of linear Diophantine equations [Dwo98]. They are also used for solving algorithmic problems in lattice based theory [HM89]. In 2001, Micciancio [Mic01] suggested the use of HNF matrices for improving the security and efficiency of GGH [GGH97]. In [PSW08] and [RPS11], HNF matrices form the public key to a digital signature scheme similar to that of GGH, except that the geometric distance between points in the lattice are calculated differently.

Definition 2.1 *An $m \times n$ matrix $H = (h_{i,j})$ is in Hermite Normal Form if there exists $r \leq m$, and a strictly increasing map $f : [r+1] \to [1, m]$ satisfying the following properties.*

1. *For $r + 1 \leq j \leq m, h_{f(j),j} \geq 1, h_{i,j} = 0$ if $i > f(j)$ and $0 \leq h_{f(k),j} < h_{f(k),k}$ if $k < j$.*
2. *The last $m - r$ rows of H are equal to 0.*

More generally, if $m \geq n$, a matrix $H \in \mathbb{Z}^{m \times n}$ in Hermite Normal Form has the following shape,

$$\begin{pmatrix} * & * & \dots & * \\ 0 & * & \dots & * \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & * \\ 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 0 & 0 \end{pmatrix}$$

where the first m rows form a matrix in HNF.

In the special case where $m = n$ and $f(k) = k$, H is in Hermite Normal Form if it satisfies the following conditions.

1. H is an upper triangle matrix, i.e. $h_{i,j} = 0$ if $i > j$.
2. For every i, $h_{i,i} > 0$.
3. For every $i > j$, we have $0 \leq h_{i,j} < h_{i,i}$.

For example, the following matrix is in Hermite Normal Form.

$$H = \begin{bmatrix} 2 & 0 & 1 & 11 \\ 0 & 1 & 5 & 12 \\ 0 & 0 & 6 & 12 \\ 0 & 0 & 0 & 33 \end{bmatrix}$$

Notice that all off diagonal entries in the lower half of the matrix are zero (condition 1); all diagonal entries are greater than zero (condition 2); and all off diagonal entries in the upper half of the matrix are greater than or equal to zero, but less than the value of the entry on the diagonal (condition 3).

Theorem 2.1 *Let A be an $m \times n$ matrix with coefficients in $\mathbb{Z}$. Then, there exists a unique $m \times n$ matrix $H = (h_{i,j})$ in HNF of the form $H = UA$ with $U \in \mathrm{GL}_m(\mathbb{Z})$, where $\mathrm{GL}_m(\mathbb{Z})$ is the general linear group of matrices with integer coefficients which are invertible, and whose determinant is equal to ± 1. Such matrices are commonly referred as being* unimodular.

Proof: The existence statement of Theorem 2.1 is proved by [Coh93], while the uniqueness statement is proved by [MG02]. □

Note that, although H is unique, the matrix U will *not* be unique. Further note that the non-zero rows of H form the HNF of the matrix A. Algorithm 1 illustrates the traditional method of calculating HNF using Gaussian elimination; the only difference is that there is no last column.

This algorithm terminates since one can easily prove that $|a_{i,k}|$ is strictly decreasing with each iteration of the first loop. Also, it is clear that H is the HNF of A, since it has been obtained from A using elementary row operations of determinant ± 1.

Example. Suppose one is given the following matrix.

$$A = \begin{pmatrix} 0 & -1 & 1 & 0 \\ -2 & 3 & 2 & 1 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix}$$

On the first iteration of the first loop, row 2 is found to contain the smallest absolute coefficient in the first column of the matrix. This is often referred to as the pivot entry. It is therefore swapped with row 1 of the matrix giving

$$A^{(j \leftarrow 1)}_{(\text{first loop})} = \begin{pmatrix} -2 & 3 & 2 & 1 \\ 0 & -1 & 1 & 0 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix}$$

Since $a_{1,1}$ is less than zero, the entire row is negated giving

$$A^{(j \leftarrow 1)}_{(\text{first loop})} = \begin{pmatrix} 2 & -3 & -2 & -1 \\ 0 & -1 & 1 & 0 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix}$$

Each row below the first row is then reduced based on the pivot entry (in this case, the first coefficient of the matrix) resulting in the following matrix

$$A^{(j \leftarrow 1)}_{(\text{first loop})} = \begin{pmatrix} 2 & -3 & -2 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & -10 & -8 & -3 \\ 0 & 4 & 2 & 12 \end{pmatrix}$$

Algorithm 1 Compute Hermite Normal Form

Given an $m \times n$ matrix A with integer coefficients $(a_{i,j})$ the following algorithm finds the HNF H of A. As usual, $h_{i,j}$ is written for the coefficients of H, A_i (resp. H_i) for the rows of A (resp. H).

Input: An $m \times n$ matrix A with integer coefficients $(a_{i,j})$.
Output: The HNF H of A.

$k \leftarrow 1$;
for $j \leftarrow 1$ **to** n **do**
  // Choose Pivot
  $i_0 \leftarrow \{i \in [k,n] : a_{i,j} = \min\{|a_{i,j}| : i \in [k,n]\}\}$;
  if $i_0 > k$ **then**
    Exchange row A_k with row A_{i_0};
    if $a_{k,j} < 0$ **then**
      $A_k \leftarrow -A_k$;
    end if
  end if
  // Reduce Rows
  $b \leftarrow a_{k,j}$;
  for $i \leftarrow (k+1)$ **to** n **do**
    $q \leftarrow \lfloor a_{i,j}/b \rceil$;
    $A_i \leftarrow A_i - q \times A_k$;
  end for
  $k \leftarrow k+1$;
end for

// Final Reductions
$k \leftarrow 1$;
for $j \leftarrow 1$ **to** n **do**
  if $a_{k,j} < 0$ **then**
    $A_k \leftarrow -A_k$;
  end if
  $b \leftarrow a_{k,j}$;
  if $b = 0$ **then**
    continue;
  else
    for $i \leftarrow 1$ **to** $(k-1)$ **do**
      $q \leftarrow \lfloor a_{i,j}/b \rfloor$;
      $A_i \leftarrow A_i - q \times A_k$;
    end for
  end if
  $k \leftarrow k+1$;
end for

for $i \leftarrow 1$ **to** n **do**
  $H_i \leftarrow A_i$;
end for

Reducing a row involves eliminating the coefficients below the pivot entry by means of elementary row operations (in this case, by some multiple of the first row). As such, they do not change the solution set of the system of linear equations represented by the matrix.

On the second iteration of the first loop, row 2 is found to contain the pivot entry (that is, the smallest absolute coefficient, this time in the second column of the matrix). It, therefore, does not need to be swapped. It does, however, need to be negated, although this is deferred till the second loop of the algorithm. The remaining rows are, again, reduced based on the pivot entry, resulting in the following matrix

$$A^{(j\leftarrow 2)}_{(\text{first loop})} = \begin{pmatrix} 2 & -3 & -2 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -18 & -3 \\ 0 & 0 & 6 & 12 \end{pmatrix}$$

On the third iteration of the first loop, row 4 is found to contain the pivot entry (that is, the smallest absolute coefficient, this time in the third column of the matrix). It is, therefore, swapped with row 3 of the matrix giving

$$A^{(j\leftarrow 3)}_{(\text{first loop})} = \begin{pmatrix} 2 & -3 & -2 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 6 & 12 \\ 0 & 0 & -18 & -3 \end{pmatrix}$$

Note that, since the pivot entry is already positive, the row does not need to be negated. The remaining row, however, is reduced based on the pivot entry giving rise to the following matrix

$$A^{(j\leftarrow 3)}_{(\text{first loop})} = \begin{pmatrix} 2 & -3 & -2 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 6 & 12 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

Having completed the first loop, the second loop of the algorithm then reduces the coefficients in the upper triangular portion of the matrix, only this time making them consistent with Definition 2.1. It turns out that this forms the uniqueness of the HNF matrix. Since this reduction is based on the pivot entry on the diagonal, the first iteration of the second loop will have no effect on the matrix. On the second

iteration of the loop, however, the first row of the matrix is reduced giving

$$A^{(j\leftarrow 2)}_{\text{(second loop)}} = \begin{pmatrix} 2 & 0 & -5 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 6 & 12 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

Notice that the step to negate the second row of the matrix is performed prior to the reduction step, and was combined with the above result. This ensures that all coefficients on the diagonal remain positive in line with the definition. On the third iteration of the second loop, both rows 1 and 2 are reduced giving

$$A^{(j\leftarrow 3)}_{\text{(second loop)}} = \begin{pmatrix} 2 & 0 & 1 & 11 \\ 0 & 1 & 5 & 12 \\ 0 & 0 & 6 & 12 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

As the matrix is now in HNF format, the final iteration of the second loop will have no effect. Thus,

$$H = \begin{pmatrix} 2 & 0 & 1 & 11 \\ 0 & 1 & 5 & 12 \\ 0 & 0 & 6 & 12 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

Note that the final reduction steps could have easily been placed inside the first loop, thereby avoiding the second loop, and making the implementation more efficient. In this and subsequent algorithms, the final reduction steps are kept separate for clarity of implementation, but also to demonstrate their independence to other row reduction steps.

Remarks.

1. In the case where A is of equal rank (i.e. $m = n$), it is easy to modify the above algorithm (as well as subsequent ones) so as to give a lower triangular HNF of A.

2. If the matrix $U \in \mathrm{GL}_n(\mathbb{Z})$ is required, it is easy to add the corresponding statements to calculate it (see Algorithm 2.4.10 in [Coh93]).

2.2 Hermite Normal Form using GCD

Consider the simple case where $m = 2, n = 1$ of this algorithm. The result will typically be a 1×1 matrix whose unique element is equal to the GCD of $a_{1,1}$ and $a_{2,1}$. Hence, it is usually faster to replace the divisions of Algorithm 1 with (extended) GCD's. This gives rise to Algorithm 2.

Example. Suppose one is given the same matrix as the previous example,

$$A = \begin{pmatrix} 0 & -1 & 1 & 0 \\ -2 & 3 & 2 & 1 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix}$$

On the first iteration of the first (outer) loop, the Euclidean step is used to determine the GCD between rows 1 and 2 of the matrix. Unlike Algorithm 1, there is no pivot entry, and therefore no need to swap the rows. Instead, the GCD of $a_{1,1}$ and $a_{2,1}$ is computed with $d = 2$, $u = 0$ and $v = -1$. These values are first used to compute the auxiliary vector B, which will be the final result (in this case) of the first row.

$$B = (2, -3, -2, -1)$$

These values are then used to reduce the second row of the matrix based on a multiple of the entries in the first row.

$$A_2 = (0, -1, 1, 0)$$

Finally, the auxiliary vector is assigned to the first row of the matrix giving

$$A^{(j \leftarrow 1, i \leftarrow 2)}_{(\text{first loop})} = \begin{pmatrix} 2 & -3 & -2 & -1 \\ 0 & -1 & 1 & 0 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix}$$

The Euclidean step is then used to determine the GCD between rows 1 and 3 of the matrix. This time, the GCD of $a_{1,1}$ and $a_{3,1}$ is computed with $d = 2$, $u = 1$ and $v = 0$. This does not change the auxiliary vector, however, the third row of the matrix is reduced to

$$A_3 = (0, -10, -8, -3).$$

Algorithm 2 Compute Hermite Normal Form using GCD

Given an $m \times n$ matrix A with integer coefficients $(a_{i,j})$ this algorithm finds the HNF H of A, using an auxiliary row vector B.

Input: An $m \times n$ matrix A with integer coefficients $(a_{i,j})$.
Data: Auxiliary row vector B.
Output: The HNF H of A.

```
k ← 1;
for j ← 1 to n do
  for i ← (k + 1) to n do
    if a_{i,j} ≠ 0 then
      // Euclidean Step
      Compute (d, u, v) such that u a_{k,j} + v a_{i,j} = d = gcd(a_{k,j}, a_{i,j});
      B ← u A_k + v A_i;
      A_i ← (a_{k,j}/d) A_i − (a_{i,j}/d) A_k;
      A_k ← B;
    end if
  end for
  k ← k + 1;
end for

// Final Reductions
k ← 1;
for j ← 1 to n do
  if a_{k,j} < 0 then
    A_k ← −A_k;
  end if
  b ← a_{k,j};
  if b = 0 then
    continue;
  else
    for i ← 1 to (k − 1) do
      q ← ⌊a_{i,j}/b⌋;
      A_i ← A_i − q × A_k;
    end for
  end if
  k ← k + 1;
end for

for i ← 1 to n do
  H_i ← A_i;
end for
```

This results in the following matrix

$$A^{(j\leftarrow 1, i\leftarrow 3)}_{\text{(first loop)}} = \begin{pmatrix} 2 & -3 & -2 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & -10 & -8 & -3 \\ 4 & -2 & -2 & 10 \end{pmatrix}.$$

Applying the same process to rows 1 and 4 of the matrix results in the following matrix

$$A^{(j\leftarrow 1, i\leftarrow 4)}_{\text{(first loop)}} = \begin{pmatrix} 2 & -3 & -2 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & -10 & -8 & -3 \\ 0 & 4 & 2 & 12 \end{pmatrix}.$$

On the second iteration of the first (outer) loop, the Euclidean step is used to determine the GCD between rows 2 and 3 of the matrix. In this case, the GCD of $a_{2,2}$ and $a_{3,2}$ is computed with $d = 1$, $u = -1$ and $v = 0$. These values are first used to compute the auxiliary vector B, which will be the final result (in this case) of the second row.

$$B = (0, 1, -1, 0)$$

These values are then used to reduce the third row of the matrix based on a multiple of the entries in the second row.

$$A_3 = (0, 0, 18, 3)$$

Finally, the auxiliary vector is assigned to the second row of the matrix giving

$$A^{(j\leftarrow 2, i\leftarrow 3)}_{\text{(first loop)}} = \begin{pmatrix} 2 & -3 & -2 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 18 & 3 \\ 0 & 4 & 2 & 12 \end{pmatrix}$$

The Euclidean step is then used to determine the GCD between rows 2 and 4 of the matrix. This time, the GCD of $a_{2,2}$ and $a_{4,2}$ is computed with $d = 1$, $u = 1$ and $v = 0$. This does not change the auxiliary vector, however, the fourth row of the matrix is reduced to

$$A_4 = (0, 0, 6, 12).$$

This results in the following matrix

$$A^{(j\leftarrow 2, i\leftarrow 4)}_{\text{(first loop)}} = \begin{pmatrix} 2 & -3 & -2 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 18 & 3 \\ 0 & 0 & 6 & 12 \end{pmatrix}.$$

On the third iteration of the first (outer) loop, the Euclidean step is used to determine the GCD between rows 3 and 4 of the matrix. In this case, the GCD of $a_{3,3}$ and $a_{4,3}$ is computed with $d = 6$, $u = 0$ and $v = 1$. These values are first used to compute the auxiliary vector B, which will be the final result (in this case) of the third row.

$$B = (0, 0, 6, 12)$$

These values are then used to reduce the fourth row of the matrix based on a multiple of the entries in the third row.

$$A_4 = (0, 0, 0, 33)$$

Finally, the auxiliary vector is assigned to the third row of the matrix giving

$$A^{(j\leftarrow 3, i\leftarrow 4)}_{\text{(first loop)}} = \begin{pmatrix} 2 & -3 & -2 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 6 & 12 \\ 0 & 0 & 0 & 33 \end{pmatrix}.$$

Note that the second loop of the algorithm is identical to that of Algorithm 1. As such, its output will be no different to that of Algorithm 1. It reduces the coefficients in the upper triangular portion of the matrix, making them consistent with Definition 2.1. This ultimately results in the following matrix

$$H = \begin{pmatrix} 2 & 0 & 1 & 11 \\ 0 & 1 & 5 & 12 \\ 0 & 0 & 6 & 12 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

which is in HNF format. Like Algorithm 1, the final reduction steps could have easily been placed inside the first loop. Like before, they were kept separate for clarity of implementation, but also to demonstrate their independence to other row reduction steps.

Remark. Cohen [Coh93] suggests that u and v be chosen between a certain range in order to avoid an infinite loop. The above adaptation of Cohen's algorithm uses structured programming techniques, which avoids infinite loops, regardless of how u and v are selected. Furthermore, most implementations of Euclid's extended algorithm calculate u and v to be consistent with the bound noted by Cohen. According to Cohen, such a pair exists and is unique.

2.3 Hermite Normal Form using Modulo D

Note that Algorithms 1 and 2 work entirely with integers, and there are no divisions except for those of the Euclidean steps. One would therefore expect that they behave reasonably well with respect to the size of the integers involved. Unfortunately, this is not entirely the case, since they fail to bind the coefficients of intermediate results, even in very reasonable circumstances. For instance, in [HM90], Hefner and McCurley provide an example of a 20×20 integer matrix whose coefficients are less than or equal to 10, but need integers of up to 1500 decimal digits to perform the calculation.

One improvement to Algorithm 2 would be to fix column j and, instead of performing operations between rows i and j, perform the operations between rows k and $k-1$, then $k-1$ and $k-2$, and so forth until rows 2 and 1 are reached. Then, exchange rows 1 and k. This idea was contributed by Bradley [Bra71].

Yet another modification is to perform row operations in the following order: $(k, k-1), (k, k-2), \ldots, (k, 1)$. For every column j, one can also work with the pair of rows (i_1, i_2) where $a_{i_1,j}$ and $a_{i_2,j}$ are the largest and second largest non-zero elements of column j with $i \leq k$. Experiments show that the coefficients of intermediate results are considerably reduced, as is the computational time of preceding versions [Coh93].

When $m \leq n$ and A is of rank m (in which case H is an upper triangular matrix with non-zero determinant D), an important improvement suggested by several authors is to work modulo the determinant of H, or even modulo a multiple of the determinant of H.

In the case where $m = n$, the $\det(H) = \pm\det(A)$. Therefore, the determinant can be computed before doing the reduction if required. In general, however, one does not know $\det(H)$ in advance. In practice, however, the HNF is often used for obtaining a basis for a lattice $\mathcal{L}$, in which case one typically knows a multiple of

the determinant of $\mathcal{L}$. This gives rise to Algorithm 3 contributed by Hafner and McCurley [HM91].

Algorithm 3 COMPUTE HERMITE NORMAL FORM MODULO D

Let A be an $m \times n$ integer matrix of rank n. Let $L = (l_{i,j})_{1 \le i,j \le n}$ be the $n \times n$ upper triangular matrix obtained from A by doing all operations modulo D in any of the above mentioned algorithms, where D is a positive multiple of the determinant of the subspace generated by the columns of A (or equivalently, the determinant of the HNF of A). This algorithm outputs the true upper triangular HNF $H = (h_{i,j})_{1 \le i,j \le n}$ of A. Also, let H_i and L_i denote the i-th rows of H and L respectively.

Input: An $n \times n$ matrix L with integer coefficients $(l_{i,j})$.
Output: The HNF H of L.

```
b ← det(A);
for i ← 1 to n do
    // Euclidean Step
    Compute (d, u, v) such that ul_{i,i} + vb = d = gcd(l_{i,i}, b);
    H_i ← uL_i  mod |b|;
    if d = |b| then
        h_{i,i} ← d;
    end if
    b ← b/d;
end for

for j ← 2 to n do
    for i ← 1 to (j − 1) do
        q ← ⌊h_{i,j}/h_{j,j}⌋;
        H_i ← H_i − q × H_j;
    end for
end for
```

To prove that this algorithm is valid note that, since the first loop is executed exactly n times, the algorithm terminates. Therefore, the only thing to prove is that the matrix H is indeed the HNF of A. For any $m \times n$ matrix M of rank n, let $\gamma_i(M)$ denote the GCD of all the $i \times i$ sub-determinants (or *leading principal minors*) obtained from the first i rows of M for $1 \le i \le n$. It is clear that elementary row operations like those of Algorithms 1 and 2 leave these quantities unchanged. Furthermore, reduction modulo D changes these $i \times i$ leading principal minors by multiples of D, and hence does not change the GCD of $\gamma_i(M)$ and D. It is also clear

that $\gamma_{n-i+1}(H) = h_{i,i} \cdots h_{n,n}$ divides $\det(H)$, and hence divides D. Therefore,

$$\begin{aligned} h_{i,i} \cdots h_{n,n} &= \gcd(D, \gamma_{n-i+1}(H)) \\ &= \gcd(D, \gamma_{n-i+1}(A)) \\ &= \gcd(D, \gamma_{n-i+1}(L)) \\ &= \gcd(D, l_{i,i} \cdots l_{n,n}). \end{aligned} \tag{2.1}$$

Hence, the value given by Algorithm 3 for $h_{n,n}$ is correct.

Let D_i denote the value of b at the i-th iteration (that is, the $i \times i$ leading principal minor of matrix A), and set $P_i = h_{i+1,i+1} \cdots h_{n,n}$. Then, assuming that the diagonal elements of $h_{j,j}$ are correct for $j > i$, by definition $D_i = D/P_i$. Hence, if equation 2.1 is divided by P_i, then

$$1 = \gcd(D_i, (l_{i+1,i+1} \cdots l_{n,n})/P_i)$$

for $1 \leq i < n$. Now, by the preceding formula,

$$h_{i,i} = \gcd(D_i, (l_{i,i} \cdots l_{n,n})/P_i) = \gcd(D_i, l_{i,i}).$$

Hence, the diagonal elements of the matrix H which are output by Algorithm 3 are correct. Since H is an upper triangular matrix, it follows that its determinant is equal to the determinant of the HNF of A. It remains to show that the rows $H_i = (uL_i \mod D_i)$ output by the algorithm are in the lattice $\mathcal{L}$ generated by the rows of A. See [Coh93] for further details.

Example. To fully demonstrate the capability of this algorithm, Algorithm 2 is first modified to perform row operations modulus the determinant of the input matrix A. The steps that will be specifically affected include the calculation of the auxiliary vector, as well as the row reduction of the lower triangular portion of the matrix.

$$\begin{aligned} B &\leftarrow uA_k + vA_i \pmod{|\det A|}; \\ A_i &\leftarrow (a_{k,j}/d)A_i - (a_{i,j}/d)A_k \pmod{|\det A|}; \\ A_k &\leftarrow B \pmod{|\det A|}; \end{aligned}$$

Also affected is the final reduction step which reduces the upper triangular portion of the matrix.

$$A_i \leftarrow A_i - q \times A_k \pmod{|\det A|};$$

Rather than using the same matrix as the previous example as input, though, consider the following matrix,

$$A = \begin{pmatrix} 1 & 2 & -4 & 1 \\ 2 & -5 & -3 & -3 \\ -6 & -3 & 0 & 5 \\ -6 & 7 & 0 & -8 \end{pmatrix}$$

Using the modified version of Algorithm 2 to determine the HNF of A results in the following matrix,

$$L = \begin{pmatrix} 1 & 0 & 0 & 335 \\ 0 & 1 & 2 & 739 \\ 0 & 0 & 3 & 2116 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

which is an upper triangular matrix in HNF format, but not quite the HNF of A. To correct this, Algorithm 3 is used.

On the first iteration of the first loop, the Euclidean step is used to determine the GCD between $l_{1,1}$ and the determinant of the matrix denoted by b. This results in $d = 1$, $u = 1$ and $v = 0$. These values are used to compute the first row of H, modulo the determinant of A.

$$H_1 = (1, 0, 0, 335)$$

This results in the following matrix,

$$H^{(i \leftarrow 1)}_{(\text{first loop})} = \begin{pmatrix} 1 & 0 & 0 & 335 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The determinant of the matrix denoted by b is then factored using the GCD of $l_{1,1}$ and b as the divisor. Since $d = \gcd(l_{1,1}, b) = 1$, this results in $b = -2873$.

On the second iteration of the first loop, the Euclidean step is used to determine the GCD between $l_{2,2}$ and b. This results in $d = 1$, $u = 1$ and $v = 0$. These values are used to compute the second row of H, modulo the factored determinant of A denoted by b.

$$H_2 = (0, 1, 2, 739)$$

This results in the following matrix,

$$H^{(i\leftarrow 2)}_{(\text{first loop})} = \begin{pmatrix} 1 & 0 & 0 & 335 \\ 0 & 1 & 2 & 739 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The determinant of the matrix denoted by b is then factored using the GCD of $l_{2,2}$ and b as the divisor. Since $d = \gcd(l_{2,2}, b) = 1$, this results in $b = -2873$.

On the third iteration of the first loop, the Euclidean step is used to determine the GCD between $l_{3,3}$ and b. This results in $d = 1$, $u = 958$ and $v = 1$. These values are used to compute the third row of H, modulo the factored determinant of A denoted by b.

$$H_3 = (0, 0, 1, 1663)$$

This results in the following matrix,

$$H^{(i\leftarrow 3)}_{(\text{first loop})} = \begin{pmatrix} 1 & 0 & 0 & 335 \\ 0 & 1 & 2 & 739 \\ 0 & 0 & 1 & 1663 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The determinant of the matrix denoted by b is then factored using the GCD of $l_{2,2}$ and b as the divisor. Since $d = \gcd(l_{3,3}, b) = 1$, this results in $b = -2873$.

On the fourth iteration of the first loop, the Euclidean step is used to determine the GCD between $l_{4,4}$ and b. This results in $d = 2873$, $u = 0$ and $v = -1$. These values are used to compute the fourth row of H, modulo the factored determinant of A denoted by b.

$$H_4 = (0, 0, 0, 0)$$

But, since d is now equal to $|b|$, $h_{4,4}$ is assigned the value of d, resulting in the following matrix,

$$H^{(i\leftarrow 4)}_{(\text{first loop})} = \begin{pmatrix} 1 & 0 & 0 & 335 \\ 0 & 1 & 2 & 739 \\ 0 & 0 & 1 & 1663 \\ 0 & 0 & 0 & 2873 \end{pmatrix}$$

Notice how the resulting matrix is closer to the true HNF of A. It, however, is not in HNF format. To fix this, the second loop of Algorithm 3 which reduces the upper triangular portion of the matrix is used.

On the first iteration of the (outer) loop, row 1 is reduced by a multiple of row 2 based on the diagonal entry. Specifically, a factor between $h_{1,2}$ and $h_{2,2}$ is chosen. In this case, since $h_{1,2} = 0$, q will be equal to zero, and therefore no change to the first row will occur.

$$H^{(j\leftarrow 2)}_{\text{(second loop)}} = \begin{pmatrix} 1 & 0 & 0 & 335 \\ 0 & 1 & 2 & 739 \\ 0 & 0 & 1 & 1663 \\ 0 & 0 & 0 & 2873 \end{pmatrix}$$

On the second iteration of the (outer) loop, row 1 is first reduced by a multiple of row 3, only this time a factor between $h_{1,3}$ and $h_{3,3}$ is chosen. Since $h_{1,3} = 0$, q will be equal to zero and, once again, no change to the first row will occur. Row 2 is then reduced by a multiple of row 3, choosing a factor between $h_{2,3}$ and $h_{3,3}$. Since $h_{2,3} = 2$, q will be assigned a value of 2, resulting in the second row of the matrix being reduced to

$$H_2 = (0, 1, 0, -2587).$$

This results in the following matrix,

$$H^{(j\leftarrow 3)}_{\text{(second loop)}} = \begin{pmatrix} 1 & 0 & 0 & 335 \\ 0 & 1 & 0 & -2587 \\ 0 & 0 & 1 & 1663 \\ 0 & 0 & 0 & 2873 \end{pmatrix}$$

On the third iteration of the (outer) loop, rows 1, 2 and 3 are reduced by a multiple of row 4, each time choosing factors between $h_{1,4}$ and $h_{4,4}$, $h_{2,4}$ and $h_{4,4}$, and $h_{3,4}$ and $h_{4,4}$, respectively. Reducing row 1 by a multiple of row 4 will not have any effect since $h_{4,4}$ does not divide $h_{1,4}$. This is also true for reducing row 3 by a multiple of row 4. Reducing row 2 by a multiple of row 4, however, will result in a factor of -1 being calculated for q, thereby reducing the second row of the matrix to

$$H_2 = (0, 1, 0, 286).$$

This results in the following matrix, which is the true HNF of A.

$$H = \begin{pmatrix} 1 & 0 & 0 & 335 \\ 0 & 1 & 0 & 286 \\ 0 & 0 & 1 & 1663 \\ 0 & 0 & 0 & 2873 \end{pmatrix}$$

Remark. In the case where the modified version of Algorithm 2 returns a true HNF matrix, applying Algorithm 3 to it will have no effect. Such is the case for the example of the previous section. Knowing when not to apply Algorithm 3 will therefore depend on the preceding algorithm's ability to correctly calculate the HNF of a matrix.

2.4 Hermite Normal Form using Modulo D and GCD

Note that, if modulo D is applied to the results of the column operations of Algorithm 2, the order in which the rows are treated is not important. Furthermore, the proof of Algorithm 3 shows that it is not necessary to apply modulo computation to the full multiple of the determinant D in Algorithm 2, but that at column i one can apply modulo a multiple of D (or D_i), which turns out to be much smaller. Finally note that, in the first loop of Algorithm 3, if modulo D (or D_i) is applied, it may happen that $h_{i,i} = 0$. In that case, it is necessary to set $h_{i,i} \leftarrow D_i$ (or any non-zero multiple of D_i). Combining these observations leads to Algorithm 4, essentially contributed by Domich *et al.* [DKT87].

Example. Suppose one is given the same matrix as previous examples,

$$A = \begin{pmatrix} 0 & -1 & 1 & 0 \\ -2 & 3 & 2 & 1 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix}$$

Since $a_{1,1} = 0$, on the first iteration of first (outer) loop, $a_{1,1}$ is assigned the value of the determinant denoted by R, resulting in the following matrix.

$$A^{(j\leftarrow 1, i\leftarrow 1, k\leftarrow 1)}_{(\text{first loop})} = \begin{pmatrix} 396 & -1 & 1 & 0 \\ -2 & 3 & 2 & 1 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix}$$

The Euclidean step is then used to determine the GCD between $a_{1,1}$ and itself. This is valid, since it should not affect the values of the current row. Theoretically, it

Algorithm 4 COMPUTE HERMITE NORMAL FORM MODULO D

Given an $m \times n$ matrix A with integer coefficients $(a_{i,j})$ of rank m (such that $m \leq n$), and a positive integer D which is known to be a multiple of the determinant of the subspace generated by the rows of A, this algorithm finds the HNF H of A, using an auxiliary row vector B.

Input: An $m \times n$ matrix A with integer coefficients $(a_{i,j})$.
Data: An auxiliary row vector B.
Output: The HNF H of L.

$D \leftarrow \det(A)$;
$R \leftarrow D$;
$k \leftarrow 1$;
for $j \leftarrow 1$ **to** n **do**
  for $i \leftarrow k$ **to** n **do**
    if $a_{k,j} = 0$ **then**
      $a_{k,j} \leftarrow R$;
    end if
    // Euclidean Step
    Compute (d, u, v) such that $ua_{k,j} + va_{i,j} = d = \gcd(a_{k,j}, a_{i,j})$;
    $B \leftarrow uA_k + vA_i$;
    $A_i \leftarrow (a_{k,j}/d)A_i - (a_{i,j}/d)A_k \mod |R|$;
    $A_k \leftarrow B \mod |R|$;
  end for
  $R \leftarrow R/d$;
  $k \leftarrow k + 1$;
end for

$k \leftarrow 1$;
$R \leftarrow D$;
for $j \leftarrow 1$ **to** n **do**
  Compute (d, u, v) such that $ua_{k,j} + vR = d = \gcd(a_{k,j}, R)$;
  $H_j \leftarrow uA_k \mod |R|$;
  if $h_{j,j} = 0$ **then**
    $h_{j,j} \leftarrow |R|$;
  end if
  for $i \leftarrow 1$ **to** $(k - 1)$ **do**
    $q \leftarrow \lfloor h_{i,j}/h_{j,j} \rfloor$;
    $H_i \leftarrow H_i - q \times H_k$;
  end for
  $R \leftarrow R/d$;
  $k \leftarrow k + 1$;
end for

should provide the same solution. It turns out that $d = 396$, $u = 0$ and $v = 1$, which results in the following auxiliary row vector being computed.

$$B = (396, -1, 1, 0)$$

Notice that this is identical to the first row of the matrix. Further notice that, although the first row will be reduced modulo the determinant of the matrix, in this case, it is overwritten by auxiliary vector, which itself is modulo reduced to

$$B = (0, 395, 1, 0)$$

This results in the following matrix.

$$A^{(j\leftarrow 1, i\leftarrow 1, k\leftarrow 1)}_{\text{(first loop)}} = \begin{pmatrix} 0 & 395 & 1 & 0 \\ -2 & 3 & 2 & 1 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix}$$

Since $a_{1,1} = 0$, it is again assigned the value of the determinant denoted by R, resulting in the following matrix.

$$A^{(j\leftarrow 1, i\leftarrow 2, k\leftarrow 1)}_{\text{(first loop)}} = \begin{pmatrix} 396 & 395 & 1 & 0 \\ -2 & 3 & 2 & 1 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix}$$

The Euclidean step is then used to determine the GCD between rows 1 and 2 of the matrix. That is, between $a_{1,1}$ and $a_{2,1}$. This gives $d = 2$, $u = 0$ and $v = -1$, which results in the following auxiliary row vector being computed.

$$B = (2, -3, -2, -1)$$

The second row is then reduced by a multiple of the first row modulo the determinant of the matrix denoted by R giving,

$$A_2 = (0, 197, 1, 198)$$

The auxiliary vector is then assigned to the first row of the matrix giving,

$$A^{(j\leftarrow 1, i\leftarrow 2, k\leftarrow 1)}_{\text{(first loop)}} = \begin{pmatrix} 2 & -3 & -2 & -1 \\ 0 & 197 & 1 & 198 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix}$$

The Euclidean step is then used to determine the GCD between rows 1 and 3 of the matrix. That is, between $a_{1,1}$ and $a_{3,1}$. This gives $d = 2$, $u = 1$ and $v = 0$, which results in the following auxiliary row vector being computed.

$$B = (2, 393, 394, 395)$$

The third row is then reduced by a multiple of the first row modulo the determinant of the matrix denoted by R giving,

$$A_3 = (0, 386, 388, 393)$$

The auxiliary vector is then assigned to the first row of the matrix giving,

$$A^{(j\leftarrow 1, i\leftarrow 3, k\leftarrow 1)}_{(\text{first loop})} = \begin{pmatrix} 2 & 393 & 394 & 393 \\ 0 & 197 & 1 & 198 \\ 0 & 386 & 388 & 393 \\ 4 & -2 & -2 & 10 \end{pmatrix}$$

The Euclidean step is then used to determine the GCD between rows 1 and 4 of the matrix. That is, between $a_{1,1}$ and $a_{4,1}$. This gives $d = 2$, $u = 1$ and $v = 0$, which results in the following auxiliary row vector being computed.

$$B = (2, 393, 394, 395)$$

The fourth row is then reduced by a multiple of the first row modulo the determinant of the matrix denoted by R giving,

$$A_4 = (0, 4, 2, 12)$$

The auxiliary vector is then assigned to the first row of the matrix giving,

$$A^{(j\leftarrow 1, i\leftarrow 4, k\leftarrow 1)}_{(\text{first loop})} = \begin{pmatrix} 2 & 393 & 394 & 393 \\ 0 & 197 & 1 & 198 \\ 0 & 386 & 388 & 393 \\ 0 & 4 & 2 & 12 \end{pmatrix}$$

Prior to the second iteration of the first (outer) loop, the determinant of the matrix is factored by the last value of d, which in this case is 2, giving a new determinant factor of $R = 198$. This is used in the next iteration of the loop to determine the GCD between $a_{2,2}$ and itself. As mentioned earlier, this should not

affect the values of the current row. It turns out that $d = 197$, $u = 0$ and $v = 1$, which results in the following auxiliary row vector being computed.

$$B = (0, 197, 1, 198)$$

Notice that this identical to the second row of the matrix. Further notice that, although the second row will be reduced modulo the new determinant factor of the matrix, it is overwritten by the auxiliary vector, which itself is modulo reduced to

$$B = (0, 197, 1, 0)$$

This results in the following matrix.

$$A^{(j\leftarrow 2, i\leftarrow 2, k\leftarrow 2)}_{(\text{first loop})} = \begin{pmatrix} 2 & 393 & 394 & 393 \\ 0 & 197 & 1 & 0 \\ 0 & 386 & 388 & 393 \\ 0 & 4 & 2 & 12 \end{pmatrix}$$

The Euclidean step is then used to determine the GCD between rows 2 and 3 of the matrix. That is, between $a_{2,2}$ and $a_{3,2}$. This gives $d = 1$, $u = 145$ and $v = -74$, which results in the following auxiliary row vector being computed.

$$B = (0, 1, -28567, -29082)$$

This is modulo reduced to

$$B = (0, 1, 143, 24)$$

The third row is then reduced by a multiple of the second row modulo the new determinant factor of the matrix denoted by R giving,

$$A_3 = (0, 0, 18, 3)$$

The auxiliary vector is then assigned to the second row of the matrix giving,

$$A^{(j\leftarrow 2, i\leftarrow 3, k\leftarrow 2)}_{(\text{first loop})} = \begin{pmatrix} 2 & 393 & 394 & 393 \\ 0 & 1 & 143 & 24 \\ 0 & 0 & 18 & 3 \\ 0 & 4 & 2 & 12 \end{pmatrix}$$

The Euclidean step is finally used to determine the GCD between rows 2 and 4 of the matrix. That is, between $a_{2,2}$ and $a_{4,2}$. This gives $d = 1$, $u = 1$ and $v = 0$, which results in the following auxiliary row vector being computed.

$$B = (0, 1, 143, 24)$$

The fourth row is then reduced by a multiple of the second row modulo the new determinant factor of the matrix denoted by R giving,

$$A_4 = (0, 0, 24, 114)$$

The auxiliary vector is then assigned to the second row of the matrix giving,

$$A^{(j\leftarrow 2, i\leftarrow 4, k\leftarrow 2)}_{(\text{first loop})} = \begin{pmatrix} 2 & 393 & 394 & 393 \\ 0 & 1 & 143 & 24 \\ 0 & 0 & 18 & 3 \\ 0 & 0 & 24 & 114 \end{pmatrix}$$

Prior to the third iteration of the first (outer) loop, the determinant of the matrix is factored by the last value of d, which in this case is 1, therefore not changing the new determinant factor denoted by R. The same value is therefore used in the next iteration of the loop to determine the GCD between $a_{3,3}$ and itself. It turns out that $d = 18$, $u = 0$ and $v = 1$, which results in the following auxiliary row vector being computed[1].

$$B = (0, 0, 18, 3)$$

As the reduction of the third row is overwritten by the assignment of the auxiliary vector, there is no change in the matrix. The change occurs when the Euclidean step is used to determine the GCD between rows 3 and 4 of the matrix. That is, between $a_{3,3}$ and $a_{4,3}$. This gives $d = 6$, $u = -1$ and $v = 1$, which results in the following auxiliary row vector being computed.

$$B = (0, 0, 6, 111)$$

The fourth row is then reduced by a multiple of the third row modulo the new determinant factor of the matrix denoted by R giving,

$$A_4 = (0, 0, 0, 132)$$

The auxiliary vector is then assigned to the third row of the matrix giving,

$$A^{(j\leftarrow 3, i\leftarrow 4, k\leftarrow 3)}_{(\text{first loop})} = \begin{pmatrix} 2 & 393 & 394 & 393 \\ 0 & 1 & 143 & 24 \\ 0 & 0 & 6 & 111 \\ 0 & 0 & 0 & 132 \end{pmatrix}$$

[1] I'll give a bottle of Scotch Whiskey to anybody who reads this line.

Prior to the fourth iteration of the first (outer) loop, the determinant of the matrix is factored by the last value of d, which in this case is 6, giving a new determinant factor of $R = 33$. This value is used in the next iteration of the loop to determine the GCD between $a_{4,4}$ and itself. It turns out that $d = 132$, $u = 0$ and $v = 1$, which results in the following auxiliary row vector being computed.

$$B = (0, 0, 0, 132)$$

As the reduction of the fourth row is overwritten by the assignment of the auxiliary vector, there is no change in the matrix. However, since $R = 33$ divides $a_{4,4} = 132$, the modulo reduction causes the fourth row of the matrix to be cleared. This results in the following matrix.

$$A^{(j\leftarrow 4, i\leftarrow 4, k\leftarrow 4)}_{(\text{first loop})} = \begin{pmatrix} 2 & 393 & 394 & 395 \\ 0 & 1 & 143 & 24 \\ 0 & 0 & 6 & 111 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

At this point, Algorithm 3 may be used to return the true HNF of the above matrix, since this is what the second loop of Algorithm 4 is based on. What distinguishes the second loop of Algorithm 4 from Algorithm 3 above is the fact that it combines the reduction of the upper triangular portion of the matrix with other steps for calculating the HNF, including the Euclidean step.

Prior to entering the second loop, R is re-initialised to the determinant of A, or to a factor thereof. Upon entering the loop, the Euclidean step is used to determine the GCD between $a_{1,1}$ and the determinant of the matrix denoted by R, resulting in $d = 2$, $u = 1$ and $v = 0$. The first row of the true HNF matrix is then calculated and reduced modulo R, resulting in the following matrix.

$$H^{(j\leftarrow 1, k\leftarrow 1)}_{(\text{second loop})} = \begin{pmatrix} 2 & 393 & 394 & 395 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Prior to entering the second iteration of the loop, R is factored by the last value of d, which in this case is 2, giving a new determinant factor of $R = 198$. This value is used in the next iteration of the loop to determine the GCD between $a_{2,2}$ and R, resulting in $d = 1$, $u = 1$ and $v = 0$. The second row of the true HNF matrix is then

calculated and reduced modulo R, resulting in the following matrix.

$$H^{(j\leftarrow 2,k\leftarrow 2)}_{\text{(second loop)}} = \begin{pmatrix} 2 & 393 & 394 & 393 \\ 0 & 1 & 143 & 24 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The first row is further reduced by a multiple of the second row modulo the new determinant factor denoted by R. This results in the following matrix.

$$H^{(j\leftarrow 2,k\leftarrow 2)}_{\text{(second loop)}} = \begin{pmatrix} 2 & 0 & 31 & 71 \\ 0 & 1 & 143 & 24 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Prior to entering the third iteration of the loop, R is factored by the last value of d, which in this case is 1, therefore not changing the new determinant factor of $R = 198$. This value is therefore used again in the next iteration of the loop to determine the GCD between $a_{3,3}$ and R, resulting in $d = 6$, $u = 1$ and $v = 0$. The third row of the true HNF matrix is then calculated and reduced modulo R, resulting in the following matrix.

$$H^{(j\leftarrow 3,k\leftarrow 3)}_{\text{(second loop)}} = \begin{pmatrix} 2 & 0 & 31 & 71 \\ 0 & 1 & 143 & 24 \\ 0 & 0 & 6 & 111 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The first and second rows are further reduced by a multiple of the third row modulo the new determinant factor denoted by R. This results in the following matrix.

$$H^{(j\leftarrow 3,k\leftarrow 3)}_{\text{(second loop)}} = \begin{pmatrix} 2 & 0 & 1 & 110 \\ 0 & 1 & 5 & 45 \\ 0 & 0 & 6 & 111 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Prior to entering the fourth iteration of the loop, R is factored by the last value of d, which in this case is 6, giving a new determinant factor of $R = 33$. This value is used in the next iteration of the loop to determine the GCD between $a_{4,4}$ and R, resulting in $d = 33$, $u = 0$ and $v = 1$. The fourth row of the true HNF matrix is

then calculated and reduced modulo R, resulting in the following matrix.

$$H^{(j\leftarrow 4,k\leftarrow 4)}_{\text{(second loop)}} = \begin{pmatrix} 2 & 0 & 1 & 110 \\ 0 & 1 & 5 & 45 \\ 0 & 0 & 6 & 111 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Since $h_{4,4}$ is now equal to zero, it is assigned the new determinant factor denoted by R. The first, second and third rows are then reduced by a multiple of the fourth row modulo R. This results in the following HNF matrix.

$$H = \begin{pmatrix} 2 & 0 & 1 & 11 \\ 0 & 1 & 5 & 12 \\ 0 & 0 & 6 & 12 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

Remark. In the case where the first loop returns a true HNF matrix, applying the second loop will have no effect. As noted earlier, the second loop could have easily been replaced with Algorithm 3.

2.5 Hermite Normal Form using Chinese Remainder Theorem

In the case of square integer matrices $A \in \mathbb{Z}^{n\times n}$, the determinant $d = \det A$ can be computed efficiently by calculating its residue modulo sufficiently many distinct primes, and then recovering the final result using the Chinese Remainder Theorem [vzGG99]. This is done by choosing distinct primes that are guaranteed to be greater than $2|d|$, perform Gaussian elimination on $A \mod p \in \mathbb{Z}_p^{n\times n}$ to calculate $d \mod p$, and represent this value between

$$-\frac{p-1}{2}, \ldots, \frac{p-1}{2}. \tag{2.2}$$

The calculation of $\det(A \mod p)$ is essentially the same as Gaussian elimination over $\mathbb{Q}$, except that when dividing by a pivot element a, one has to calculate its inverse modulo p using the extended Euclidean algorithm. What is gained is a reduction in size of the intermediate values, since they are bound between the range noted by (2.2). It remains to determine a "good" a priori bound on $|\det A|$; one

that is only polynomially large in n and the size of the coefficients of A, and which is easy to determine without actually calculating $\det A$. Such a bound is provided by the Hadamard inequality [MW01]. Combining this observation with the previous observation concerning the GCD leads to Algorithm 5, essentially contributed by Micciancio and Warinschi [MW01].

Algorithm 5 COMPUTE HERMITE NORMAL FORM USING CRT

Given an $n \times n$ matrix A with integer coefficients $(a_{i,j})$ of rank n, and a set of distinct prime numbers which are at least two times the size of the Hadamard bound, this algorithm finds the HNF H of A using auxiliary column vectors $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$, and auxiliary matrices B and H_B. It also returns the determinant of H.

Input: An $m \times n$ matrix A with integer coefficients $(a_{i,j})$, and a set of distinct prime numbers which are at least two times the size of the Hadamard bound.
Data: Two auxiliary row vectors $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$, and auxiliary matrices B and H_B.
Output: The HNF H of A.

$h_{1,1} \leftarrow a_{1,1}$
for $i \leftarrow 2$ **to** n **do**
  $\hat{\mathbf{a}}^T(i-1) \leftarrow (a_{1,i}, a_{2,i}, \ldots, a_{i,i})$;
  $\hat{\mathbf{x}}^T(i-1) \leftarrow \text{ADDCOLUMN}(A(i-1), H(i-1), \hat{\mathbf{a}}^T(i-1))$;
  $H(i) \leftarrow \text{ADDROW}([H(i-1) \;\; \hat{\mathbf{x}}^T(i-1)], (a_{i,1}, \ldots, a_{i,i}))$;
end for
return $H(n)$;

The idea behind this method of calculating the HNF is to decompose the matrix, A, into:

$$A = \left[\begin{array}{c|c} B & \hat{\mathbf{a}}^T \\ \hline \hat{\mathbf{b}} & \end{array}\right]$$

where $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$ are row vectors. Then,

1. recursively compute the HNF H_B of B,
2. extend H_B to the HNF H' of $B' = \left[\begin{array}{cc} B & \hat{\mathbf{a}}^T \end{array}\right]$,
3. finally, compute and return the HNF of $\left[\begin{array}{c} H' \\ \hat{\mathbf{b}} \end{array}\right]$.

In order to execute steps (2) and (3) above, the following two procedures are required.

- ADDCOLUMN$(B, H_B, \hat{\mathbf{a}}^T)$: inputs a square non-singular matrix B, its HNF H_B, and a column vector $\hat{\mathbf{a}}^T$; returns an unique column vector $\hat{\mathbf{x}}^T$ such that $H' = \left[\begin{array}{cc} H_B & \hat{\mathbf{x}}^T \end{array}\right]$ is the HNF of $B' = \left[\begin{array}{cc} B & \hat{\mathbf{a}}^T \end{array}\right]$.

- AddRow($H', \hat{\mathbf{b}}$): inputs a matrix H' in HNF, and a row vector $\hat{\mathbf{b}}$; returns the HNF of matrix $\begin{bmatrix} H' \\ \hat{\mathbf{b}} \end{bmatrix}$.

Together, they form the foundation for building the HNF matrix column by row until the desired dimension of the target matrix is reached[2]. These are elaborated in further detail in the sections below.

The AddColumn procedure

The AddColumn procedure takes as its input a non-singular matrix, B, its HNF matrix, H_B, and a column vector $\hat{\mathbf{a}}^T$. Its output is a vector, $\hat{\mathbf{x}}^T$, such that

$$H_A = \begin{bmatrix} H_B & \hat{\mathbf{x}}^T \end{bmatrix}$$

is the HNF of

$$A = \begin{bmatrix} B & \hat{\mathbf{a}}^T \end{bmatrix}.$$

Note that, if U is a unique unimodular transformation such that $H_B = UB$ then $\hat{\mathbf{x}}^T$ is simply $U\hat{\mathbf{a}}^T = H_B B^{-1}\hat{\mathbf{a}}^T \Rightarrow U = H_B B^{-1}$. In this case, the entries of U can be as large as the determinant of A, not making it practical to be stored at the same time. Instead, $\hat{\mathbf{x}}$ can be indirectly calculated as follows: For a suitably chosen sequence of primes, $p_1, p_2, ...$

- compute a solution $\hat{\mathbf{y}}_i$ to the system of equations $B\hat{\mathbf{y}}_i^T = \hat{\mathbf{a}}^T \pmod{p_i}$,
- compute $\hat{\mathbf{x}}_i^T = H_B\hat{\mathbf{y}}_i^T \pmod{p_i}$.

For sufficiently many primes, p_i, $\hat{\mathbf{x}}^T$ can then be recovered using the Chinese Remainder Theorem.

In order to bind the number of primes necessary to correctly recover $\hat{\mathbf{x}}$, one needs to bind the entries of $\hat{\mathbf{x}}^T$. This also binds both the time and space complexity of AddColumn. Let M be an upper bound to the absolute value of the elements in B, and let $h_1, h_2, \ldots, h_m$ be the diagonal elements of B (in particular, $\prod_i h_i = D$, and one can safely assume that $\sum_{i=1}^{n} h_i \leq D$). Since the entries of B^{-1} are bounded by $D = \det(B)$, an element of $H_B B^{-1}$ is $O(\sum_{j=1}^{n} Dh_j) \leq O(D^2)$. Therefore, an entry of $\hat{\mathbf{x}}$ has an upper bound $V = O(nMD^2) = O(nM^{2n+1})$. For simplicity, it is

[2]As noted earlier, Micciancio and Warinschi use lower triangular matrices to represent their HNFs. As such, they define their matrices using column vectors. This means that the AddColumn procedure defined in this paper should be substituted for the AddRow procedure, and vice-versa.

assumed that $D = O(M^n)$; a more accurate bound is the Hadamard bound. The bit size of $\hat{\mathbf{x}}$ is thus $O(n^2 \log M)$, and a rough estimate of the number of primes needed to recover $\hat{\mathbf{x}}$ is $O(\log V)$. By the following proposition, proved in [KB79], the largest of these is $\log V \log\log V$.

Proposition 2.2 *If p_n denotes the n-th prime number, then $p_n = O(n \log n)$.*

Each of the system of equations modulo p_i can be solved in $O(n^3 \log^2 p_i)$ by Gaussian elimination. So, AddColumn is $O(\log V)O(n^3 \log^2(\log V \log\log V))$ which, after expanding V becomes $O^{\sim}(n^4 \log^2(nM))$. Faster methods for solving systems of linear equations are described in [Dix82] and [MS99] based on p-adic expansion. It is plausible that the same techniques can be applied to the implementation of AddColumn, reducing the running time from $O(n^4 \log^2(nM))$ to $n^3 \log^2(nM)$ [MW01].

The AddRow procedure

The AddRow procedure takes as input a matrix $H' \in \mathbb{Z}^{(n-1)\times n}$ in Hermite Normal Form, and a row vector $\hat{\mathbf{b}} \in \mathbb{Z}^n$. Its output is the Hermite Normal Form $H \in \mathbb{Z}^{n\times n}$ of $\begin{bmatrix} H' \\ \hat{\mathbf{b}} \end{bmatrix}$. The procedure works by first extending H' to a square matrix $H_0 = \begin{bmatrix} H' \\ \hat{\mathbf{c}} \end{bmatrix}$ in Hermite Normal form such that $\begin{bmatrix} H_0 \\ \hat{\mathbf{b}} \end{bmatrix}$ generates the same lattice as $\begin{bmatrix} H' \\ \hat{\mathbf{b}} \end{bmatrix}$. This is simply done by setting $\hat{\mathbf{c}} = (0, \ldots, d)$, where d is the determinant of $\begin{bmatrix} H' \\ \hat{\mathbf{b}} \end{bmatrix}$. A sequence of matrix-vector pairs, $H_j, \hat{\mathbf{b}}_j$, for $j = 0, ..., n$, are then computed such that,

- $\hat{\mathbf{b}}_0 = \hat{\mathbf{b}}$,
- H_i is in Hermite Normal Form,
- the first i elements of $\hat{\mathbf{b}}$ are 0.

Each $H_{i+1}, \hat{\mathbf{b}}_{i+1}$ pair is obtained from the previous $H_i, \hat{\mathbf{b}}_i$ pair as follows. If the $(i+1)$th element of $\hat{\mathbf{b}}$ is zero, then $H_{i+1} = H_i$ and $\hat{\mathbf{b}}_{i+1} = \hat{\mathbf{b}}_i$ is simply set. Otherwise, the $(i+1)$th row of H_i and $\hat{\mathbf{b}}_i$ are replaced with two other rows obtained by applying a unimodular transformation that clears the (i+1)th element of $\hat{\mathbf{b}}_i$. This is done by executing the extended Euclidean algorithm on the last elements of rows H_{i+1} and

$\hat{\mathbf{b}}_i$. Once this is done, the remaining elements of the two rows might be greater than the diagonal elements of H_i. If so, the columns are reduced modulo the diagonal elements of H_i using the last i rows of H_i.

Algorithm 6 AddRow

Input: A matrix $H' \in \mathbb{Z}^{(i-1)\times i}$ in Hermite Normal Form, and a column vector $\hat{\mathbf{b}} \in \mathbb{Z}^i$, for each iteration, i, in the main loop of the algorithm.

Output: The Hermite Normal Form $H \in \mathbb{Z}^{i\times i}$ of $\begin{bmatrix} H' \\ \hat{\mathbf{b}} \end{bmatrix}$.

Set H to the matrix $\begin{bmatrix} H' \\ \hat{\mathbf{c}} \end{bmatrix}$ where $\hat{\mathbf{c}} = \left[0, \ldots, 0, \det \begin{bmatrix} H' \\ \hat{\mathbf{b}} \end{bmatrix}\right]$;
$m_n \leftarrow h_{n,n}$;
for $i = n$ **downto** 1 **do**
  $m_i = m_{i+1} \cdot h_{i,i}$;
end for
for $i = 1$ **to** n **do**
  find k, l, g such that $kh_{i,i} + lb_i = g = \gcd(h_{i,i}, b_i)$;
  for $j = i$ **to** n **do**
    $h_{j,i} \leftarrow kh_{j,i} + lb_j \pmod{m_j}$;
    $b_j \leftarrow b_j h_{i,i}/g - h_{j,i} b_i/g \pmod{m_j}$;
  end for
  for $k = i + 1$ **to** n **do**
    $q = \lfloor h_{i,k}/h_{k,k} \rfloor$;
    for $l = k$ **to** n **do**
      $h_{i,l} = h_{i,l} - qh_{k,l} \pmod{m_l}$;
    end for
  end for
end for

To analyse the space complexity of AddRow, assume that the size of the input matrix A, and consequently the size of H, is $O(n^2 \log M)$. It is apparent that this assumption holds true during the execution of Algorithm 5. For one iteration of the main loop, only the i-th row of H and $\hat{\mathbf{b}}$ are modified. The entries are also kept bounded by performing computation modulo m_j. The space therefore required by these two rows is $O(\sum_{i=1}^{n} \log m_i) = O(n \log m_1)$. Since $m_1 = \det(H)$, the space requirement becomes $O(n^2 \log M)$. And, due to the triangular reductions of the second inner loop, the matrix $\begin{bmatrix} H' \\ \hat{\mathbf{b}} \end{bmatrix}$ requires $O(n^2 \log M)$ storage space at the beginning of the next iteration. Since all computations are done in place, the total space required by AddColumn is $O(n^2 \log M)$.

The main computational part of the algorithm is the triangular reduction procedure of the second inner loop taken from [Sto96] with a running time of $O(n \log^2 D)$, where D is the determinant of the matrix A. In this case, it is assumed that D is of order $O(M^n)$ (see above), and since the execution of the first inner loop takes $O(n^2 \log^2 M)$, the combined execution time of AddColumn is $n(O(n \log^2 M^n) + O(n^2 \log^2 M)) = O(n^4 \log^2 M)$.

In order to prove that these operations do not change the lattice, one has to show that they correspond to sequences of elementary column operations. Regarding the modular reduction operations, notice that m_k is the determinant of the sub-matrix corresponding to the non-zero rows of the first k rows of $H(i)$ (for all $k > i$). So, the vector $(0, \ldots, 0, m_k, 0, \ldots, 0)$ belongs to the lattice generated by the first $k+1$ rows of $H(i)$, and reducing the k-th entry of a vector modulo m_k corresponds to subtracting appropriate multiples of the first $k+1$ rows of $H(i)$. Finally, notice that the row operations of the first inner loop corresponds to the linear transformation

$$\begin{bmatrix} u & v \\ -b_i/g & h_{i,i}/g \end{bmatrix}$$

which has a determinant equal to 1 by definition of u, v, g. So, this transformation is unimodular, and corresponds to a sequence of elementary row operations. This proves that the lattice generated by $\begin{bmatrix} H_n \\ \hat{\mathbf{b}}_n \end{bmatrix}$ is the same as the original lattice generated by $\begin{bmatrix} A \\ \hat{\mathbf{b}} \end{bmatrix}$. Moreover, H_n is the Hermite normal form and $\hat{\mathbf{b}} = \mathbf{0}$. Therefore $H = H_n$ is the Hermite normal form of $\begin{bmatrix} A \\ \hat{\mathbf{b}} \end{bmatrix}$.

Example. Suppose one is given the same matrix as the previous example,

$$A = \begin{pmatrix} 0 & -1 & 1 & 0 \\ -2 & 3 & 2 & 1 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix}$$

First, a set of distinct primes greater than two times the size of the determinant need to be chosen. As noted earlier, in order to bind the number of primes necessary to recover $\hat{\mathbf{x}}$, one needs to bind the entries in $\hat{\mathbf{x}}^T$. By noting that $M = \max\{|a| :$

$a \in A\} = 10$, one can determine the upper bound for $\hat{\mathbf{x}}^T$ as $V = O(nMD^2) = 4 \times 10 \times 396^2 = 6272640$. The number of primes to recover $\hat{\mathbf{x}}$ is therefore $O(\log V) \approx 6$.

Let P denote the set of primes used to recover $\hat{\mathbf{x}}$. Using an appropriate deterministic function to select primes greater than V, one would expect primes similar to the following set to be chosen.

$$P = \{6272659, 6272663, 6272681, 6272683, 6272689, 6272723\}$$

Prior to entering the main loop, $h_{1,1}$ is assigned the value of $a_{1,1}$ giving a single element matrix as illustrated below.

$$H = \begin{pmatrix} 0 \end{pmatrix}$$

B is also assigned the leading principal sub-matrix of A of dimension 1×1, giving a simple element matrix as illustrated below.

$$B = \begin{pmatrix} 0 \end{pmatrix}$$

For AddColumn to compute a solution $\hat{\mathbf{y}}_i$ to the system of equations $B\hat{\mathbf{y}}_i^T = \hat{\mathbf{a}}^T \pmod{P_i}$, it requires that B be invertible. This requires that its determinant be non-zero. Since its single element is zero, its determinant is zero. Therefore, the above system of equations cannot be solved.

According to [KB79], however, there exists a method to permute the rows of a non-singular matrix such that all its principal minors are non-singular. Another method is to multiply the matrix by a randomly chosen unimodular matrix, which is equivalent to performing standard row operations on a matrix. To demonstrate the latter point, consider multiplying the following unimodular matrix by A.

$$U = \begin{pmatrix} 2 & 11 & 22 & 41 \\ 5 & 31 & 70 & 123 \\ 3 & 23 & 63 & 102 \\ 1 & 7 & 18 & 30 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 2 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 1 & 3 & 2 & 1 \end{pmatrix}$$
$$\times \begin{pmatrix} 3 & 2 & 2 & 1 \\ 3 & 3 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 3 \\ 1 & 2 & 0 & 2 \end{pmatrix}$$

This gives,

$$\begin{aligned} U \times A &= \begin{pmatrix} 2 & 11 & 22 & 41 \\ 5 & 31 & 70 & 123 \\ 3 & 23 & 63 & 102 \\ 1 & 7 & 18 & 30 \end{pmatrix} \times \begin{pmatrix} 0 & -1 & 1 & 0 \\ -2 & 3 & 2 & 1 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix} \\ &= \begin{pmatrix} 10 & -73 & -102 & 421 \\ 10 & -228 & -319 & 1261 \\ -16 & -201 & -281 & 1043 \\ -2 & -58 & -81 & 307 \end{pmatrix} \end{aligned}$$

whose leading principal minors are all non-singular. To see this, consider breaking the matrix into principal sub-matrices as follows

$$(U \times A)(1) = \begin{pmatrix} 10 \end{pmatrix} \Longrightarrow \det((U \times A)(1)) = 10$$

$$(U \times A)(2) = \begin{pmatrix} 10 & -73 \\ 10 & -228 \end{pmatrix} \Longrightarrow \det((U \times A)(2)) = -1550$$

$$(U \times A)(3) = \begin{pmatrix} 10 & -73 & -102 \\ 10 & -228 & -319 \\ -16 & -201 & -281 \end{pmatrix} \Longrightarrow \det((U \times A)(3)) = -1116$$

$$(U \times A)(4) = \begin{pmatrix} 10 & -73 & -102 & 421 \\ 10 & -228 & -319 & 1261 \\ -16 & -201 & -281 & 1043 \\ -2 & -58 & -81 & 307 \end{pmatrix} \Longrightarrow \det((U \times A)(4)) = 396$$

where $(U \times A)(n)$ denotes the n-th leading principal minor of the matrix formed by $U \times A$. Further notice that the determinant of $U \times A$ is the same as the determinant of A. Its rows therefore span the same lattice as A.

Continuing with the example, $h_{1,1}$ is assigned the new value of $a_{1,1}$ giving a single element matrix as illustrated below.

$$H = \begin{pmatrix} 10 \end{pmatrix}$$

B is also assigned the leading principal sub-matrix of A of dimension 1×1, giving a single element matrix as illustrated below.

$$B = \begin{pmatrix} 10 \end{pmatrix}$$

Since this is non-singular, $\hat{\mathbf{y}}_i$ can be solved for $B\hat{\mathbf{y}}_i^T = \hat{\mathbf{a}}^T \pmod{P_i}$. For example, for $i = 1$, and $\hat{\mathbf{a}}^T = (-73)$,

$$\begin{aligned}[10]\,\hat{\mathbf{y}}_1^T &= (-73) \pmod{6272659} \\ \hat{\mathbf{y}}_1^T &= [10]^{-1}\,(-73) \pmod{6272659} \\ &= (4390854)\end{aligned}$$

For $i = 2$, and $\hat{\mathbf{a}}^T = (-73)$,

$$\begin{aligned}[10]\,\hat{\mathbf{y}}_2^T &= (-73) \pmod{6272663} \\ \hat{\mathbf{y}}_2^T &= [10]^{-1}\,(-73) \pmod{6272663} \\ &= (627259)\end{aligned}$$

etc. This, in turn, is used to solve $\hat{\mathbf{x}}_i^T = H_B\hat{\mathbf{y}}_i^T \pmod{p_i}$. Table 2.1 shows the solution of $\hat{\mathbf{y}}_i^T$ and $\hat{\mathbf{x}}_i^T$ for all primes P_i.

i	P_i	$\hat{\mathbf{y}}_i^T$	$\hat{\mathbf{x}}_i^T$
1	6272659	$(4390854)^T$	$(6272586)^T$
2	6272663	$(627259)^T$	$(6272590)^T$
3	6272681	$(1881797)^T$	$(6272608)^T$
4	6272683	$(627261)^T$	$(6272610)^T$
5	6272689	$(4390875)^T$	$(6272616)^T$
6	6272723	$(627265)^T$	$(6272650)^T$

Table 2.1: Solving for $\hat{\mathbf{y}}_i^T$ and $\hat{\mathbf{x}}_i^T$ for all primes P_i.

Using the Chinese Remainder Theorem, one can now recover $\hat{\mathbf{x}}$. In this case $\hat{\mathbf{x}} = (-73)^T$, which is the same as the input vector $\hat{\mathbf{a}}^T$. This is used to form the following intermediate matrix.

$$H' = \begin{pmatrix} 10 & -73 \end{pmatrix}$$

This, together with the following row vector are input into AddRow.

$$\hat{\mathbf{b}} = \begin{pmatrix} 10 & -228 \end{pmatrix}$$

The first step of AddRow is to set H to the matrix

$$\begin{pmatrix} H' \\ \hat{\mathbf{c}} \end{pmatrix}$$

where

$$\hat{\mathbf{c}} = \left(0, \ldots, 0, \left|\det \begin{pmatrix} H' \\ \hat{\mathbf{c}} \end{pmatrix}\right|\right).$$

In this case,

$$H = \begin{pmatrix} 10 & -73 \\ 0 & 1550 \end{pmatrix}$$

On the first iteration of the outer loop, the extended Euclidean algorithm is first used to find (g, k, l) such that $kh_{i,i} + lb_i = g = \gcd(h_{i,i}, b_i)$. In this case,

$$\begin{aligned} & kh_{1,1} + lb_1 = g = \gcd(h_{1,1}, b_1) \\ \Longrightarrow & 10k + 10l = g = \gcd(10, 10) \\ \Longrightarrow & g = 10, k = 0, l = 1 \end{aligned}$$

This is used by the inner loop to calculate the upper triangular portion of the matrix row by row. This results in the following matrix

$$H = \begin{pmatrix} 10 & -228 \\ 0 & 1550 \end{pmatrix}$$

The row vector $\hat{\mathbf{b}}$ is also reduced modulo the current determinant giving

$$\hat{\mathbf{b}} = \begin{pmatrix} 0 & 1395 \end{pmatrix}$$

This is equivalent to the initial row reductions of Algorithm 4 except that, in this case, $\hat{\mathbf{b}}$ is the auxiliary vector.

On the second iteration of the outer loop, the extended Euclidean algorithm is again used to find (g, k, l) such that $kh_{i,i} + lb_i = g = \gcd(h_{i,i}, b_i)$. In this case,

$$\begin{aligned} & kh_{2,2} + lb_2 = g = \gcd(h_{2,2}, b_2) \\ \Longrightarrow & 1550k + 1395l = g = \gcd(1550, 1395) \\ \Longrightarrow & g = 155, k = 1, l = -1 \end{aligned}$$

This results in the following matrix,

$$H = \begin{pmatrix} 10 & -228 \\ 0 & 155 \end{pmatrix}$$

The row vector $\hat{\mathbf{b}}$ is again reduced modulo the current determinant giving

$$\hat{\mathbf{b}} = \begin{pmatrix} 0 & 0 \end{pmatrix}$$

The second loop reduces the upper triangular portion of the matrix modulo the determinant row by row. In this case, it will only modify the first row of the matrix. In particular,

$$\begin{aligned} q &= \lfloor h_{1,2}/h_{2,2} \rfloor \\ &= \lfloor -228/155 \rfloor \\ &= -2 \end{aligned}$$

resulting in the following matrix,

$$H = \begin{pmatrix} 10 & 82 \\ 0 & 155 \end{pmatrix}$$

which is the true HNF of the matrix

$$(U \times A)(2) = \begin{pmatrix} 10 & -73 \\ 10 & -228 \end{pmatrix}.$$

AddColumn uses this new H and $B = (U \times A)(2)$ together with the next column $\hat{\mathbf{a}} = (\ -102 \quad -319\ )^T$ to calculate the intermediate matrix H', on the next iteration of the main loop. Since B is non-singular, $\hat{\mathbf{y}}_i$ can again be solved for $B\hat{\mathbf{y}}_i^T = \hat{\mathbf{a}}^T \pmod{P_i}$. For example, for $i = 1$,

$$\begin{pmatrix} 10 & -73 \\ 10 & -228 \end{pmatrix} \hat{\mathbf{y}}_1^T = \begin{pmatrix} -102 \\ -319 \end{pmatrix} \pmod{6272659}$$

$$\hat{\mathbf{y}}_1^T = \begin{pmatrix} 10 & -73 \\ 10 & -228 \end{pmatrix}^{-1} \begin{pmatrix} -102 \\ -319 \end{pmatrix} \pmod{6272659}$$

$$= \begin{pmatrix} 1379985 \\ 2509065 \end{pmatrix}$$

For $i = 2$,

$$\begin{pmatrix} 10 & -73 \\ 10 & -228 \end{pmatrix} \hat{\mathbf{y}}_2^T = \begin{pmatrix} -102 \\ -319 \end{pmatrix} \pmod{6272663}$$

$$\hat{\mathbf{y}}_2^T = \begin{pmatrix} 10 & -73 \\ 10 & -228 \end{pmatrix}^{-1} \begin{pmatrix} -102 \\ -319 \end{pmatrix} \pmod{6272663}$$

$$= \begin{pmatrix} 2885425 \\ 1254534 \end{pmatrix}$$

i	P_i	$\hat{\mathbf{y}}_i^T$	$\hat{\mathbf{x}}_i^T$
1	6272659	$(1379985 \quad 2509065)^T$	$(115 \quad 217)^T$
2	6272663	$(2885425 \quad 1254534)^T$	$(115 \quad 217)^T$
3	6272681	$(3638155 \quad 3763610)^T$	$(115 \quad 217)^T$
4	6272683	$(376361 \quad 1254538)^T$	$(115 \quad 217)^T$
5	6272689	$(5143605 \quad 2509077)^T$	$(115 \quad 217)^T$
6	6272723	$(1630908 \quad 1254546)^T$	$(115 \quad 217)^T$

Table 2.2: Solving for $\hat{\mathbf{y}}_i^T$ and $\hat{\mathbf{x}}_i^T$ for all primes P_i.

etc. This, in turn, is used to solve $\hat{\mathbf{x}}_i^T = H_B \hat{\mathbf{y}}_i^T \pmod{p_i}$. Table 2.2 shows the solution of $\hat{\mathbf{y}}_i^T$ and $\hat{\mathbf{x}}_i^T$ for all primes P_i. Notice how the solution for $\hat{\mathbf{x}}$ converges immediately with the first prime P_1.

Using the Chinese Remainder Theorem, one can now recover $\hat{\mathbf{x}}$. In this case $\hat{\mathbf{x}} = (115 \quad 217)^T$. This is used to form the following intermediate matrix.

$$H' = \begin{pmatrix} 10 & 82 & 115 \\ 0 & 155 & 217 \end{pmatrix}$$

This, together with the following row vector are input into AddRow.

$$\hat{\mathbf{b}} = \begin{pmatrix} -16 & -201 & -281 \end{pmatrix}$$

Recall that the first step of AddRow is to set H to the matrix

$$\begin{pmatrix} H' \\ \hat{\mathbf{c}} \end{pmatrix}$$

where

$$\hat{\mathbf{c}} = \left(0, \ldots, 0, \left|\det \begin{pmatrix} H' \\ \hat{\mathbf{c}} \end{pmatrix}\right|\right).$$

In this case,

$$H = \begin{pmatrix} 10 & 82 & 115 \\ 0 & 155 & 217 \\ 0 & 0 & 1116 \end{pmatrix}$$

On the first iteration of the outer loop, the extended Euclidean algorithm is first used to find (g, k, l) such that $kh_{i,i} + lb_i = g = \gcd(h_{i,i}, b_i)$. In this case,

$$\begin{aligned} & kh_{1,1} + lb_1 = g = \gcd(h_{1,1}, b_1) \\ \Longrightarrow & 10k - 16l = g = \gcd(10, -16) \\ \Longrightarrow & g = 2, k = -3, l = -2 \end{aligned}$$

This is used by the inner loop to calculate the upper triangular portion of the matrix row by row. This modifies the first row of the matrix giving,

$$H = \begin{pmatrix} 2 & 156 & 217 \\ 0 & 155 & 217 \\ 0 & 0 & 1116 \end{pmatrix}$$

The row vector $\hat{\mathbf{b}}$ is also reduced modulo the current determinant giving

$$\hat{\mathbf{b}} = \begin{pmatrix} 0 & 767 & 631 \end{pmatrix}$$

On the second iteration of the outer loop, the extended Euclidean algorithm is again used to find (g, k, l) such that $kh_{i,i} + lb_i = g = \gcd(h_{i,i}, b_i)$. In this case,

$$\begin{aligned} & kh_{2,2} + lb_2 = g = \gcd(h_{2,2}, b_2) \\ \Longrightarrow & 155k + 767l = g = \gcd(155, 767) \\ \Longrightarrow & g = 1, k = -287, l = 58 \end{aligned}$$

This modifies the second row of the matrix giving

$$H = \begin{pmatrix} 2 & 156 & 217 \\ 0 & 1 & -25681 \\ 0 & 0 & 1116 \end{pmatrix}$$

The row vector $\hat{\mathbf{b}}$ is again reduced modulo the current determinant giving

$$\hat{\mathbf{b}} = \begin{pmatrix} 0 & 0 & 558 \end{pmatrix}$$

On the third iteration of the outer loop, the extended Euclidean algorithm is again used to find (g, k, l) such that $kh_{i,i} + lb_i = g = \gcd(h_{i,i}, b_i)$. In this case,

$$\begin{aligned} & kh_{3,3} + lb_3 = g = \gcd(h_{3,3}, b_3) \\ \Longrightarrow & 1116k + 558l = g = \gcd(1116, 558) \\ \Longrightarrow & g = 558, k = 0, l = 1 \end{aligned}$$

This modifies the third row of the matrix giving

$$H = \begin{pmatrix} 2 & 156 & 217 \\ 0 & 1 & -25681 \\ 0 & 0 & 558 \end{pmatrix}$$

The row vector $\hat{\mathbf{b}}$ is again reduced modulo the current determinant giving

$$\hat{\mathbf{b}} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$$

The second loop reduces the upper triangular portion of the matrix modulo the determinant row by row. In this case, it will modify the first and second row of the matrix. For the first row, (and on the first iteration of the inner loop),

$$\begin{aligned} q &= \lfloor h_{1,2}/h_{2,2} \rfloor \\ &= \lfloor 156/1 \rfloor \\ &= 156 \end{aligned}$$

resulting in the following matrix,

$$H = \begin{pmatrix} 2 & 0 & 13 \\ 0 & 1 & -25681 \\ 0 & 0 & 558 \end{pmatrix}$$

On the second iteration of the inner loop,

$$\begin{aligned} q &= \lfloor h_{1,3}/h_{3,3} \rfloor \\ &= \lfloor 13/558 \rfloor \\ &= 0 \end{aligned}$$

resulting in no change to the matrix.

For the second row, (and the only iteration of the inner loop),

$$\begin{aligned} q &= \lfloor h_{2,3}/h_{3,3} \rfloor \\ &= \lfloor -25681/558 \rfloor \\ &= -47 \end{aligned}$$

resulting in following matrix,

$$H = \begin{pmatrix} 2 & 0 & 13 \\ 0 & 1 & 545 \\ 0 & 0 & 558 \end{pmatrix}$$

which is the true HNF of the matrix

$$(U \times A)(3) = \begin{pmatrix} 10 & -73 & -102 \\ 10 & -228 & -319 \\ -16 & -201 & -281 \end{pmatrix}.$$

AddColumn uses this new H and $B = (U \times A)(3)$ together with the next column $\hat{\mathbf{a}} = (\ 421 \quad 1261 \quad 1043\ )^T$ to calculate the intermediate matrix H', on the third iteration of the main loop. Since B is non-singular, $\hat{\mathbf{y}}_i$ can again be solved for $B\hat{\mathbf{y}}_i^T = \hat{\mathbf{a}}^T \pmod{P_i}$. For example, for $i = 1$,

$$\begin{pmatrix} 10 & -73 & -102 \\ 10 & -228 & -319 \\ -16 & -201 & -281 \end{pmatrix} \hat{\mathbf{y}}_1^T = \begin{pmatrix} 421 \\ 1261 \\ 1043 \end{pmatrix} \pmod{6272659}$$

$$\hat{\mathbf{y}}_1^T = \begin{pmatrix} 10 & -73 & -102 \\ 10 & -228 & -319 \\ -16 & -201 & -281 \end{pmatrix}^{-1} \begin{pmatrix} 421 \\ 1261 \\ 1043 \end{pmatrix} \pmod{6272659}$$

$$= \begin{pmatrix} 4637049 \\ 3271231 \\ 2664186 \end{pmatrix}$$

For $i = 2$,

$$\begin{pmatrix} 10 & -73 & -102 \\ 10 & -228 & -319 \\ -16 & -201 & -281 \end{pmatrix} \hat{\mathbf{y}}_2^T = \begin{pmatrix} 421 \\ 1261 \\ 1043 \end{pmatrix} \pmod{6272663}$$

$$\hat{\mathbf{y}}_2^T = \begin{pmatrix} 10 & -73 & -102 \\ 10 & -228 & -319 \\ -16 & -201 & -281 \end{pmatrix}^{-1} \begin{pmatrix} 421 \\ 1261 \\ 1043 \end{pmatrix} \pmod{6272663}$$

$$= \begin{pmatrix} 4367260 \\ 3810817 \\ 775644 \end{pmatrix}$$

etc. This, in turn, is used to solve $\hat{\mathbf{x}}_i^T = H_B\hat{\mathbf{y}}_i^T \pmod{p_i}$. Table 2.3 shows the solution of $\hat{\mathbf{y}}_i^T$ and $\hat{\mathbf{x}}_i^T$ for all primes P_i.

Using the Chinese Remainder Theorem, one can now recover $\hat{\mathbf{x}}$. In this case $\hat{\mathbf{x}} = (-97 \quad -4287 \quad -4395)^T$. This is used to form the following intermediate matrix.

$$H' = \begin{pmatrix} 2 & 0 & 13 & -97 \\ 0 & 1 & 545 & -4287 \\ 0 & 0 & 558 & -4395 \end{pmatrix}$$

This, together with the following row vector are input into AddRow.

$$\hat{\mathbf{b}} = \begin{pmatrix} -2 & -58 & -81 & 307 \end{pmatrix}$$

i	P_i	$\hat{\mathbf{y}}_i^T$	$\hat{\mathbf{x}}_i^T$
1	6272659	$(4637049 \quad 3271231 \quad 2664186)^T$	$(6272562 \quad 6268372 \quad 6268264)^T$
2	6272663	$(4367260 \quad 3810817 \quad 775644)^T$	$(6272566 \quad 6268376 \quad 6268268)^T$
3	6272681	$(5277823 \quad 1989727 \quad 4013159)^T$	$(6272584 \quad 6268394 \quad 6268286)^T$
4	6272683	$(2411276 \quad 1450142 \quad 5901710)^T$	$(6272586 \quad 6268396 \quad 6268288)^T$
5	6272689	$(5345278 \quad 1854833 \quad 4485302)^T$	$(6272592 \quad 6268402 \quad 6268294)^T$
6	6272723	$(5986069 \quad 573319 \quad 5834299)^T$	$(6272626 \quad 6268436 \quad 6268328)^T$

Table 2.3: Solving for $\hat{\mathbf{y}}_i^T$ and $\hat{\mathbf{x}}_i^T$ for all primes P_i.

As usual, the first step of AddRow is to set H to the matrix

$$\begin{pmatrix} H' \\ \hat{\mathbf{c}} \end{pmatrix}$$

where

$$\hat{\mathbf{c}} = \left(0, \ldots, 0, \left|\det \begin{pmatrix} H' \\ \hat{\mathbf{c}} \end{pmatrix}\right|\right).$$

In this case,

$$H = \begin{pmatrix} 2 & 0 & 13 & -97 \\ 0 & 1 & 545 & -4287 \\ 0 & 0 & 558 & -4395 \\ 0 & 0 & 0 & 396 \end{pmatrix}$$

On the first iteration of the outer loop, the extended Euclidean algorithm is first used to find (g, k, l) such that $kh_{i,i} + lb_i = g = \gcd(h_{i,i}, b_i)$. In this case,

$$\begin{aligned} & kh_{1,1} + lb_1 = g = \gcd(h_{1,1}, b_1) \\ \Longrightarrow & 2k - 2l = g = \gcd(2, -2) \\ \Longrightarrow & g = 2, k = 0, l = -1 \end{aligned}$$

This is used by the inner loop to calculate the upper triangular portion of the matrix row by row. This modifies the first row of the matrix giving,

$$H = \begin{pmatrix} 2 & 58 & 81 & -307 \\ 0 & 1 & 545 & -4287 \\ 0 & 0 & 558 & -4395 \\ 0 & 0 & 0 & 396 \end{pmatrix}$$

The row vector $\hat{\mathbf{b}}$ is also reduced modulo the current determinant giving

$$\hat{\mathbf{b}} = \begin{pmatrix} 0 & 338 & 328 & 210 \end{pmatrix}$$

On the second iteration of the outer loop, the extended Euclidean algorithm is again used to find (g, k, l) such that $kh_{i,i} + lb_i = g = \gcd(h_{i,i}, b_i)$. In this case,

$$\begin{aligned} & kh_{2,2} + lb_2 = g = \gcd(h_{2,2}, b_2) \\ \Longrightarrow & 1k + 338l = g = \gcd(1, 338) \\ \Longrightarrow & g = 1, k = 1, l = 0 \end{aligned}$$

This has no effect on the second row of the matrix as is illustrated below.

$$H = \begin{pmatrix} 2 & 58 & 81 & -307 \\ 0 & 1 & 545 & -4287 \\ 0 & 0 & 558 & -4395 \\ 0 & 0 & 0 & 396 \end{pmatrix}$$

The row vector $\hat{\mathbf{b}}$, however, is reduced modulo the current determinant giving

$$\hat{\mathbf{b}} = \begin{pmatrix} 0 & 0 & 258 & 252 \end{pmatrix}$$

On the third iteration of the outer loop, the extended Euclidean algorithm is again used to find (g, k, l) such that $kh_{i,i} + lb_i = g = \gcd(h_{i,i}, b_i)$. In this case,

$$\begin{aligned} & kh_{3,3} + lb_3 = g = \gcd(h_{3,3}, b_3) \\ \Longrightarrow & 558k + 258l = g = \gcd(558, 258) \\ \Longrightarrow & g = 6, k = -6, l = 13 \end{aligned}$$

This modifies the third row of the matrix giving

$$H = \begin{pmatrix} 2 & 58 & 81 & -307 \\ 0 & 1 & 545 & -4287 \\ 0 & 0 & 6 & 29646 \\ 0 & 0 & 0 & 396 \end{pmatrix}$$

The row vector $\hat{\mathbf{b}}$ is again reduced modulo the current determinant giving

$$\hat{\mathbf{b}} = \begin{pmatrix} 0 & 0 & 0 & 165 \end{pmatrix}$$

On the fourth iteration of the outer loop, the extended Euclidean algorithm is again used to find (g, k, l) such that $kh_{i,i} + lb_i = g = \gcd(h_{i,i}, b_i)$. In this case,

$$\begin{aligned} & kh_{4,4} + lb_4 = g = \gcd(h_{4,4}, b_4) \\ \Longrightarrow & 396k + 165l = g = \gcd(558, 258) \\ \Longrightarrow & g = 33, k = -2, l = 5 \end{aligned}$$

This modifies the fourth row of the matrix giving

$$H = \begin{pmatrix} 2 & 58 & 81 & -307 \\ 0 & 1 & 545 & -4287 \\ 0 & 0 & 6 & 29646 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

The row vector $\hat{\mathbf{b}}$ is again reduced modulo the current determinant giving

$$\hat{\mathbf{b}} = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$$

The second loop reduces the upper triangular portion of the matrix modulo the determinant row by row. In this case, it will modify the first, second and third row of the matrix. For the first row, (and on the first iteration of the inner loop),

$$\begin{aligned} q &= \lfloor h_{1,2}/h_{2,2} \rfloor \\ &= \lfloor 58/1 \rfloor \\ &= 58 \end{aligned}$$

resulting in the following matrix,

$$H = \begin{pmatrix} 2 & 0 & 151 & 47 \\ 0 & 1 & 545 & -4287 \\ 0 & 0 & 6 & 29646 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

For the second iteration of the inner loop,

$$\begin{aligned} q &= \lfloor h_{1,3}/h_{3,3} \rfloor \\ &= \lfloor 545/6 \rfloor \\ &= 25 \end{aligned}$$

resulting in the following matrix,

$$H = \begin{pmatrix} 2 & 0 & 1 & 209 \\ 0 & 1 & 545 & -4287 \\ 0 & 0 & 6 & 29646 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

For the third iteration of the inner loop,

$$\begin{aligned} q &= \lfloor h_{1,4}/h_{4,4} \rfloor \\ &= \lfloor 209/33 \rfloor \\ &= 6 \end{aligned}$$

resulting in the following matrix,

$$H = \begin{pmatrix} 2 & 0 & 1 & 11 \\ 0 & 1 & 545 & -4287 \\ 0 & 0 & 6 & 29646 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

For the second row, (and the first iteration of the inner loop),

$$\begin{aligned} q &= \lfloor h_{2,3}/h_{3,3} \rfloor \\ &= \lfloor 545/6 \rfloor \\ &= 90 \end{aligned}$$

resulting in the following matrix,

$$H = \begin{pmatrix} 2 & 0 & 1 & 11 \\ 0 & 1 & 5 & 177 \\ 0 & 0 & 6 & 29646 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

For the second iteration of the inner loop,

$$\begin{aligned} q &= \lfloor h_{2,4}/h_{4,4} \rfloor \\ &= \lfloor 177/33 \rfloor \\ &= 5 \end{aligned}$$

resulting in the following matrix,

$$H = \begin{pmatrix} 2 & 0 & 1 & 11 \\ 0 & 1 & 5 & 12 \\ 0 & 0 & 6 & 29646 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

For the third row, (and the only iteration of the inner loop),

$$\begin{aligned} q &= \lfloor h_{3,4}/h_{4,4} \rfloor \\ &= \lfloor 29646/33 \rfloor \\ &= 898 \end{aligned}$$

resulting in the following matrix,

$$H = \begin{pmatrix} 2 & 0 & 1 & 11 \\ 0 & 1 & 5 & 12 \\ 0 & 0 & 6 & 12 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

which is the true HNF of the matrix

$$(U \times A)(4) = \begin{pmatrix} 10 & -73 & -102 & 421 \\ 10 & -228 & -319 & 1261 \\ -16 & -201 & -281 & 1043 \\ -2 & -58 & -81 & 307 \end{pmatrix}.$$

It also happens to be the true HNF of the original matrix,

$$A = \begin{pmatrix} 0 & -1 & 1 & 0 \\ -2 & 3 & 2 & 1 \\ -6 & -1 & -2 & 0 \\ 4 & -2 & -2 & 10 \end{pmatrix}.$$

Remark. To reduce the number of primes necessary to recover $\hat{\mathbf{x}}$, one guideline is to recalculate the number and size of the primes required with each call to AddColumn. This assumes that the determinant of the principal sub matrices increases in size each time, which may not necessarily be the case as the above example demonstrates. Another method is to use the Hadamard bound to bind the determinant, and therefore the entries of $\hat{\mathbf{x}}$. In this case, the Hadamard bound is approximately 427, which does not reduce the number of primes necessary to recover $\hat{\mathbf{x}}$. As noted earlier, however, since the entries of B^{-1} are bounded by $D = \det(B)$, an element of $B^{-1}H_B \leq \det(B) \leq D^2$. If $m_{i,j}$ are entries in B, then

$$\prod^{k} m_{i,j} \geq 2D^2 \qquad \text{for some } k > 0.$$

Assuming that $m_{i,j} \geq \frac{b}{2}$ for some upper bound b,

$$\prod^{k} \frac{b}{2} \geq 2D^2 \qquad \text{also assuming that } \frac{b}{2} \geq 2D^2$$

Taking logarithms on both sides gives,

$$\begin{aligned} k \log_2 \frac{b}{2} &\geq \log_2 2D^2 && \text{since log is an increasing function} \\ k(\log_2 b - 1) &\geq 1 + 2\log_2 D && \text{using the expansion rules for logarithms} \end{aligned} \tag{2.3}$$

Note that,

$$\begin{aligned} D &\leq \prod^{n} \|\hat{\mathbf{b}}_i\| && \text{by Hadamard's Inequality (see [Coh93])} \\ &\leq \prod^{n} \sqrt{nM^2} && \text{where } M \text{ is the maximum absolute entry in } A \\ &\leq \prod^{n} \sqrt{n}M \\ &\leq n^{\frac{n}{2}} M^n \end{aligned} \tag{2.4}$$

Substituting (2.4) into (2.3) gives,

$$\begin{aligned} k(\log_2 b - 1) &\geq 1 + 2\log_2 n^{\frac{n}{2}} M^n \\ k(\log_2 b - 1) &\geq 1 + n\log_2 n + 2\log_2 M && \text{using the expansion rules for logarithms} \end{aligned}$$

This implies that the number of prime values, k, needed to recover $\hat{\mathbf{x}}$ is given by the following formula.

$$k \geq \frac{1 + n\log_2 n + 2\log_2 M}{\log_2 b - 1} \quad \text{for some bound } b$$

Notice that, the greater the bound of b, the fewer prime numbers required to recover $\hat{\mathbf{x}}$ using CRT.

2.6 Heuristic version of Hermite Normal Form Algorithm using Chinese Remainder Theorem

Micciancio and Warinschi [MW01] present an heuristic version of their algorithm which, in practice, achieves significantly better running time than the one described

in the previous section. They demonstrate that the new algorithm can reduce the running time by a factor n or even n^2, outperforming all previously known algorithms.

The idea is to decompose A into

$$A = \begin{bmatrix} B & \hat{\mathbf{b}}^T \\ \hat{\mathbf{c}} & a_{n-1,n} \\ \hat{\mathbf{d}} & a_{n,n} \end{bmatrix}$$

where $B \in \mathbb{Z}^{(n-2)\times(n-1)}$, $\hat{\mathbf{c}}, \mathbf{d} \in \hat{\mathbb{Z}}^{n-1}$ are row vectors, and $\hat{\mathbf{b}} \in \mathbb{Z}^{n-2}$ is a column vector. Then,

1. compute the determinants $d_1 = \det\left(\begin{bmatrix} B \\ \hat{\mathbf{c}} \end{bmatrix}\right)$, and $d_2 = \det\left(\begin{bmatrix} B \\ \hat{\mathbf{d}} \end{bmatrix}\right)$,
2. execute the extended Euclidean algorithm to find integers k and l such that $d = kd_1 + ld_2 = \gcd(d_1, d_2)$,
3. compute the HNF of the matrix $\begin{bmatrix} B \\ k\hat{\mathbf{c}} + l\hat{\mathbf{d}} \end{bmatrix}$,
4. compute the HNF H' of the matrix

$$\begin{bmatrix} B & \hat{\mathbf{b}}^T \\ k\hat{\mathbf{c}} + l\hat{\mathbf{d}} & ka_{n-1,n} + la_{n,n} \end{bmatrix},$$

   running AddColumn on input $\begin{bmatrix} B \\ k\hat{\mathbf{c}} + l\hat{\mathbf{d}} \end{bmatrix}$, H, and $\begin{bmatrix} \hat{\mathbf{b}}^T \\ ka_{n-1,n} + la_{n,n} \end{bmatrix}$,
5. run AddRow twice to add rows $(\hat{\mathbf{c}} \;\; a_{n,n-1})$ and $(\hat{\mathbf{d}} \;\; a_{n,n})$ back to H.

Notice how step 3 can be executed in time $O(n^3 \log^2 d)$ using the modulo determinant HNF algorithm described by Algorithm 4. Hence, a substantial reduction of the running time is obtained whenever the quantity $d = \gcd(d_1, d_2)$ is small.

According to Micciancio and Warinschi, when d is small, the running time of the algorithm is dominated by a single execution of the AddColumn procedure, as well as two executions of AddRow. Hence, the running time is $O(n^4 \log^2 M)$ [MW01].

Example. Suppose one is given the matrix used in Section 2.3.

$$A = \begin{pmatrix} 1 & 2 & -4 & 1 \\ 2 & -5 & -3 & -3 \\ -6 & -3 & 0 & 5 \\ -6 & 7 & 0 & -8 \end{pmatrix}$$

The first step requires decomposing A into the following,

$$A = \begin{pmatrix} B\left\{\boxed{\begin{matrix} 1 & 2 & -4 \\ 2 & -5 & -3 \end{matrix}}\boxed{\begin{matrix} 1 \\ -3 \end{matrix}}\right\}\hat{\mathbf{b}}^T \\ \hat{\mathbf{c}}\{\boxed{-6 \quad -3 \quad 0}\boxed{5}\}a_{n-1,n} \\ \hat{\mathbf{d}}\{\boxed{-6 \quad 7 \quad 0}\boxed{-8}\}a_{n,n} \end{pmatrix}$$

That is,

$$B = \begin{pmatrix} 1 & 2 & -4 \\ 2 & -5 & -3 \end{pmatrix}$$

is a matrix of dimension 2×3, $\hat{\mathbf{b}}^T = (1 \ \ -3)^T$ is a vector of length 2, $\hat{\mathbf{c}} = (-6 \ \ -3 \ \ 0)$ is a vector of length 3, $\hat{\mathbf{d}} = (-6 \ \ 7 \ \ 0)$ is a vector also of length 3, $a_{n-1,n} = 5$ and $a_{n,n} = -8$.

The first step requires that the following determinants be calculated.

$$\begin{aligned} d_1 &= \det\left(\begin{bmatrix} B \\ \hat{\mathbf{c}} \end{bmatrix}\right), & d_2 &= \det\left(\begin{bmatrix} B \\ \hat{\mathbf{d}} \end{bmatrix}\right) \\ &= \det\left(\begin{bmatrix} 1 & 2 & -4 \\ 2 & -5 & -3 \\ -6 & -3 & 0 \end{bmatrix}\right), & &= \det\left(\begin{bmatrix} 1 & 2 & -4 \\ 2 & -5 & -3 \\ -6 & 7 & 0 \end{bmatrix}\right) \\ &= 171, & &= 121 \end{aligned}$$

The second step requires that the extended Euclidean algorithm be executed to find integers k and l such that $kd_1 + ld_2 = g = \gcd(d_1, d_2)$. In this case,

$$\begin{aligned} & kd_1 + ld_2 = g = \gcd(d_1, d_2) \\ \Longrightarrow & 171k + 121l = g = \gcd(171, 121) \\ \Longrightarrow & g = 1, k = 46, l = -65 \end{aligned}$$

The third step requires that the HNF of the matrix $\begin{bmatrix} B \\ k\hat{\mathbf{c}} + l\hat{\mathbf{d}} \end{bmatrix}$ be computed. In this case,

$$\text{HNF}\left(\begin{bmatrix} B \\ k\hat{\mathbf{c}} + l\hat{\mathbf{d}} \end{bmatrix}\right) = \text{HNF}\left(\begin{bmatrix} 1 & 2 & -4 \\ 2 & -5 & -3 \\ 114 & -593 & 0 \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This can be computed using the space saving algorithm described in the previous section, although Micciancio and Warinschi recommend using the Domich and Kannan version described by Algorithm 4.

The fourth step requires that an intermediate matrix,

$$H' = \begin{bmatrix} B & \hat{\mathbf{b}}^T \\ k\hat{\mathbf{c}} + l\hat{\mathbf{d}} & ka_{n-1,n} + la_{n,n} \end{bmatrix},$$

be computed using AddColumn with input $\begin{bmatrix} B \\ k\hat{\mathbf{c}} + l\hat{\mathbf{d}} \end{bmatrix}$, H, and $\begin{bmatrix} \hat{\mathbf{b}}^T \\ ka_{n-1,n} + la_{n,n} \end{bmatrix}$. This results in the following matrix,

$$\begin{pmatrix} 1 & 0 & 0 & -28395 \\ 0 & 9 & 0 & -49140 \\ 0 & 0 & 1 & -9829 \end{pmatrix}$$

Notice that this consists of the HNF matrix of the previous step, and an extra column that has been added based on $\hat{\mathbf{b}}^T$, and the gcd of $a_{n-1,n}$, and $a_{n,n}$.

The fifth step requires that AddRow be executed twice to add rows $(\hat{\mathbf{c}} \;\; a_{n,n-1})$ and $(\hat{\mathbf{d}} \;\; a_{n,n})$ back to H. This results in the following matrix,

$$\begin{pmatrix} 1 & 0 & 0 & 335 \\ 0 & 1 & 0 & 286 \\ 0 & 0 & 1 & 1663 \\ 0 & 0 & 0 & 2873 \end{pmatrix}$$

which is the HNF of A. Note that this calculation is based on the determinant of A, rather than the determinant calculated by AddRow, which can often be greater than the determinant of the matrix. Therefore, as a preliminary step, one would need to calculate the determinant of A, and pass this as a parameter to AddRow.

Chapter 3

Distribution of Prime Determinants

In analytic number theory, the Prime Number Theorem describes the asymptotic distribution of prime numbers [EW05]. Informally speaking, it states that if a random integer is selected near to some large integer N, the probability that the selected integer is prime is about $1/\log(N)$, where $\log(N)$ denotes the natural logarithm of N. For example, near $N = 1000$, about one in seven numbers is prime, while near $N = 10,000,000,000$, about one in 23 numbers is prime. This chapter looks at the distribution of prime numbers, and how the theorem relates to the distribution of prime determinants of a matrix.

3.1 Prime Testing

Given a small integer, n, one can determine primality by testing for divisibility by known small primes. This can be improved by selecting only prime factors up to and including the $\sqrt{n}$, since any two factors will be the product of two numbers, at least one of which will be less than or equal to $\sqrt{n}$. This, however, becomes increasingly difficult as n grows in size. Alternatively, one can use Fermat's Little Theorem [EW05] to help determine primality. It, however, does not provide a necessary and sufficient condition for primality, just a necessary one. To see this consider $a = 3$ and $p = 91$. According to Fermat's Little Theorem,

$$3^{91-1} \equiv 3^{90} \equiv 1 \pmod{91}$$

yet, $91 = 7 \times 13$ is a composite number. Some early articles call all numbers satisfying this test *pseudoprimes*, but now the term *pseudoprime* is properly reserved for composite probable primes, as in this example.

Strictly speaking, Fermat's Little Theorem is not a primality test but rather a test for compositeness, since it does not prove the primality of a number. Instead,

if the number is not prime, the algorithm proves this in all likelihood very quickly by the square and multiply method. On the other hand, if the number happens to be prime, the algorithm merely provides evidence for its primality.

A more sophisticated method for determining primality is the Rabin-Miller test [Mil76]. Just like Fermat's Little Theorem, the Rabin-Miller test relies on an equality or set of equalities that hold true for prime values, then checks whether or not they hold true for the value in question.

Lemma 3.1 *Suppose p is an odd prime. Let $p-1 = 2^k m$ where m is odd. Also, let $1 \leq a < p$. Either,*

i) $a^m \equiv 1 \pmod{p}$ *or*

ii) one of

$$a^m, a^{2m}, a^{4m}, a^{8m}, \ldots, , a^{2^{k-1}m}$$

is congruent to $-1 \mod p$.

Proof: It is known that

$$\left(a^{2^{k-1}m}\right)^2 = a^{p-1} \equiv 1 \pmod{p}.$$

Thus $a^{2^{k-1}m} \equiv \pm 1 \pmod{p}$. If $a^{2^{k-1}m} \equiv -1 \pmod{p}$, then the proof is complete. Otherwise, by induction, if each of

$$a^{2^{i+1}m}, \ldots, a^{2^{k-1}m}$$

is congruent to 1, then $a^{2^i m} \equiv \pm 1 \pmod{p}$. It follows that if step *(ii)* fails, then $a^m \equiv 1 \pmod{p}$.

Suppose that there is a need to determine whether or not a given odd number n is prime. One could select $1 \leq a < n$ and calculate

$$a^m, a^{2m}, a^{4m}, a^{8m}, \ldots, , a^{2^{k-1}m} \pmod{n}.$$

If neither steps (i) nor (ii) holds true, then n is considered to be composite. In this case, it is said that a is a *witness* to n being composite. If a is not a witness, this does not imply that n is prime, but provides some evidence that n might be prime. If n is composite, most $a \in [0, n)$ will witness that it is composite. □

Theorem 3.2 (Rabin-Miller Composite Test) *If n is composite, then at least 75% of the numbers $1 < a < n$ will witness that n is composite.*

Proof: See [HPS08].

This gives rise to the following probabilistic algorithm for testing primality.

Rabin-Miller Algorithm

- Randomly pick $a_1, \ldots, a_k$ independent elements $1 < a < n$.
- For each a_i, do the test described by Lemma 3.1.
- If any a_i is a witness to n being composite, then n is composite.
- If no a_i is a witness, then guess that n is prime.

If n is determined to be composite, then there's absolute certainty that the answer is correct. If n is determined to be prime, there is some chance that the answer is incorrect. Specifically, if one were to guess that n is prime, the chance of the answer being incorrect is $(0.25)^k$, where k denotes the number of independent elements chosen. For example, if $k = 100$ elements were chosen, then the chances of the answer being incorrect is $(0.25)^{100} < 10^{-60}$. Increasing the number of elements therefore further increases the level of certainty.

As explained by Miller, though, under the Generalised Riemann Hypothesis (see [HPS08]), one can turn the Miller-Rabin algorithm into one that is deterministic in polynomial time [Mil76]. The next result gives a necessary and sufficient condition for primality. It is known as Wilson's Theorem because of a remark to this effect allegedly made by John Wilson in 1770 to the mathematician Edward Waring.

Theorem 3.3 (Wilson's Theorem) *An integer $n > 1$ is prime if and only if*

$$(n-1)! \equiv -1 \pmod{n}.$$

Proof: **[of *only if* direction]** Assume that n is an odd prime. Note that the congruence is clear for $n = 2$. Further note that each of the integers $1 < a < n-1$ has a unique multiplicative inverse distinct from $a \mod n$. For distinctness, note that $a^2 \equiv 1 \mod n$ implies that $n|(a+1)(a-1)$, forcing $a \equiv \pm 1 \mod n$ by primality. Thus, in the product

$$(n-1)! = (n-1)(n-2)\cdots 3 \cdot 2 \cdot 1,$$

all the terms cancel out modulo n, except the first and last. Their product is clearly $-1 \mod n$. □

Proof: [**of *if* direction**] Assume that $n > 1$ and that $(n-1)! \equiv -1 \mod n$. Note that the congruence is clear for $n = 2$ and $n = 3$, which are both prime. The argument is by contradiction, so suppose that n is composite. Then, its positive divisors are among the integers

$$1, 2, 3, 4, \ldots, n-1$$

It is clear that the $\gcd((n-1)!, n) > 1$, thus $(n-1)! = -1 \pmod n$ does not hold. Therefore, n must be prime. □

Note that this result is of mostly theoretical value since it is relatively difficult to calculate $(n-1)!$. In contrast, it is easy to calculate a^{n-1}, so elementary primality tests are built using Fermat's Little Theorem rather than Wilson's Theorem.

3.2 Prime Distribution

The Prime Number Theorem describes the asymptotic distribution of prime numbers. It was first proved independently in 1896 by two mathematicians - Hadamard [Had96] and de la Valée Poussin [dlVP96]. Their proofs used the Riemann zeta function and they were able to give an estimate for the error term in the formula, based upon an estimate for a zero-free region of the zeta function (See [EW05] for further details).

Let $\pi(x)$ be the prime counting function that returns the number of primes less than or equal to x, for any real number $x > 0$. For example, $\pi(10) = 4$, since the prime numbers less than or equal to 10 are 2, 3, 5 and 7.

Theorem 3.4 (Prime Number Theorem) *Asymptotically, the number of primes p less than or equal to x is given by*

$$\pi(x) = |\{p \in \mathbb{P} | p \leq x\}| \sim \frac{x}{\log x}$$

That is, the limit of the quotient of the two functions $\pi(x)$ and $x/\log(x)$ as x approaches infinity is 1. This is expressed by the following formula.

$$\lim_{x \to \infty} \frac{\pi(x)}{x/\log(x)} = 1$$

This theorem does not say anything about the limit of the difference of the two functions as x approaches infinity. Indeed, the behaviour of this difference is very

complicated, and related to the Riemann hypothesis [EW05]. Instead, the theorem states that $x/\log(x)$ approximates $\pi(x)$ in the sense that the relative error of this approximation approaches zero as x approaches infinity.

The prime number theorem is also equivalent to the statement that the n-th prime number p_n is approximately equal to $n\log(n)$, again with the relative error of this approximation approaching zero as n approaches infinity.

Intuitively, the Prime Number Theorem states that the proportion of prime numbers between 1 and x is approximately $1/\log(x)$. Turning this statement around, the Prime Number Theorem states that a chosen number x has a probability $1/\log(x)$ of being prime. For example, near $x = 1000$, about one in seven numbers is prime (14.29%), while near $x = 10,000,000,000$, about one in 23 numbers is prime (4.35%).

Around the beginning of the nineteenth century, Legendre published a conjecture equivalent to the Prime Number Theorem (see [Wei]). Gauss also studied the values of $\pi(x)$ at a similar time, and conjectured that an even better approximation is given by the logarithmic integral function $\mathrm{Li}(x)$, defined by

$$\pi(x) \sim \mathrm{Li}(x) = \int_2^x \frac{1}{\log t} dt.$$

Indeed, this integral strongly suggests that the 'density' of primes around t should be $1/\log(t)$. For small values of x, $\pi(x) < \mathrm{Li}(x)$. Several prominent mathematicians conjectured that the inequality always holds. However, in 1914 Littlewood [Lit14] proved that the inequality reverses infinitely often. Astonishingly, the smallest value of x where the inequality first reverses is still unknown. It is, however, known to be below 10^{371} (see [EW05] for further details).

In 1837, Dirichlet [Dir37] famously proved that there are infinitely many primes $p \equiv a \pmod{q}$ if and only if $\gcd(a, q) = 1$. The key ingredient was Dirichlet's L-functions $L(s, \chi)$ to study arithmetic progressions. He provides an asymptotic result that can be expressed in the following form.

$$\pi(x; a, q) = |\{q \in \mathbb{P} : p \leq x, p \equiv a \pmod{q}\}| \sim \frac{x}{\varphi(q)\log(x)} \tag{3.1}$$

where φ is Euler's totient function. This formula holds for each given progression as $x \to \infty$ and one interested in estimates uniform with q as large as possible. The Generalised Riemann Hypothesis allows q to be as large as $x^{1/2-\varepsilon}$, but all that is known unconditionally is the much smaller range $q < (\log x)^A$ for some arbitrary constant A, known as the Siegel-Walfisz theorem. On the other hand, one expects

(3.1) to hold with q as large as $x^{1-\varepsilon}$, although it is known to fail in the range $q < x(\log x)^{-A}$ for every A (see [FG89] for further details).

Viggo Brun [Bru16, Bru19, Bru22] first showed how to improve on the Legendre method by relaxing the asymptotic requirements. Instead, he concentrated on inequalities of the prime distribution, giving rise to a powerful sieve method which helped prove the following upper bound,

$$\pi(x; a, q) < \frac{cx}{\phi(q)\log(x/q)}$$

where c is an absolute constant which yields the correct order of magnitude throughout the range $q < x^{1-\varepsilon}$. This is known as the Brun-Titchmarsh theorem due to the work of Titchmarsh [Tit30]. Progress in the sieve also led to the result of $c = 2 + \varepsilon$, which can now be obtained in one of several ways (e.g. Selberg sieve, combinatorial sieve, large sieve). This is the limit of the method in several respects. An improvement in the constant c for small q would have striking consequences for the problem of exceptional zeros of L-functions, and hence for class numbers. Moreover, it is known that there are sequences of integers satisfying the same standard sieve axioms for which the corresponding bound cannot be improved. For example, the set of those integers in progression composed of an odd number of prime factors.

In view of this limitation, Y. Motohashi [Mot75] was able to improve this constant to $c = 2$ by using non-trivial information about the nature of arithmetic progressions. This work has had a significant impact on subsequent developments in general sieve theory.

3.3 Simple Prime Sieve

In terms of searching for 1024 bit primes, (i.e. primes that are approximately 2^{1024}), the theorem says that the probability of randomly choosing such a number is approximately 0.14%. Rather than selecting a number completely at random, though, one could restrict attention (say) to numbers that are relatively prime to 2, 3, 5, 7 and 11. That is, ignore all numbers that are even, all numbers that are divisible by 3, all numbers that are divisible by 5, etc., from the search. To do this, a simple sieve method described in [HPS08] is used. It first chooses a random number that is relatively prime to $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 = 2310$, say 1139, for example. Then, it considers only numbers N of the form

$$N = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot K + 1139 = 2310K + 1139 \tag{3.2}$$

The probability that an N is prime is approximately

$$\frac{2}{1} \cdot \frac{3}{2} \cdot \frac{5}{4} \cdot \frac{7}{6} \cdot \frac{11}{10} \cdot \frac{1}{\log(N)} \approx \frac{4.8}{\log(N)}.$$

That is, if one chooses a random number N of the form indicated by equation (3.2) with $N \approx 2^{1024}$, then the probability that it is prime is approximately 0.67%. Therefore, only 150 numbers need be checked in order to find one that is prime.

3.4 Distribution of Prime Determinants

Just like determining the distribution of prime numbers up to and including a certain bound is useful, so too is the distribution of prime determinants of matrices up to and including a certain dimension. In terms of discrete random matrices, Maples [Map10] proposed an inequality similar to the Brun-Titchmarsh theorem to determine a bound on the number of determinants that are prime. Noting that the Brun-Titchmarsh theorem is a simple consequence of the large sieve inequality, he uses a probabilistic analogue of the large sieve developed by Kowalski [Kow08] to prove the following theorem.

Theorem 3.5 *For all n sufficiently large, let A be a randomly chosen $n \times n$ matrix with entries taken from a distribution that is not concentrated on an arithmetic progression. Then,*

$$\text{Prob}(\det A \in \mathbb{P} \text{ and } \equiv a \mod q) \lesssim \frac{1}{\varphi(q)} \frac{1}{n}$$

for all positive integers $q \leq O(e^{cn})$ and invertible residues $a \in (\mathbb{Z}/q\mathbb{Z})^{\times}$.

In other words, the probability of the determinant of A being prime is roughly one in n, the dimension of the matrix.

Kowalski developed the following probabilistic analogue of the large sieve in [Kow08]. Let $X_1, \ldots, X_n$ denote independent, identically distributed copies of a Bernoulli random variable $X = \pm 1$ with $\text{Prob}(X = +1) = \text{Prob}(X = -1) = 1/2$. Next, define the random sum $S_n = X_1 + \cdots + X_n$. One can then view S_n as a probabilistic model for a random integer in the interval $[-\sqrt{n}, \sqrt{n}]$. With his probabilistic sieve, Kowalski proved an analogue of the Brun-Titchmarsh theory for this random sum.

Theorem 3.6 (Kowalski) *For all $q \leq x^{1/4-\varepsilon}$ and invertible residues $a \in (\mathbb{Z}/q\mathbb{Z})^\times$,*

$$\text{Prob}(S_n \in \mathbb{P} and \equiv a \mod q) \lesssim \frac{1}{\varphi(q)} \frac{1}{\log(n)}$$

where the implied constant depends only on ε.

Since $|S_n| \sim \sqrt{n}$ with high probability, it is reasonable to expect this theorem to hold for all $q \leq x^{1/2-\varepsilon}$, as predicted in [Kow08].

3.5 Minors and Determinants

Following are some fundamental concepts, definitions and notation, taken from linear algebra. Although not related to prime determinants, its inclusion here is primarily for understanding the next chapter.

A *minor* of a matrix A is the determinant of a smaller square matrix, deduced from A by removing one or more of its rows or columns. Minors obtained by removing just one row and one column from square matrices are called *first minors*. Minors obtained by removing two rows and two columns from square matrices are called *second minors*, etc. For example, given the following matrix,

$$\begin{pmatrix} 1 & 4 & 7 \\ 3 & 0 & 5 \\ -1 & 9 & 11 \end{pmatrix}$$

the first minor obtained by deleting row 2 and column 3 is calculated as follows,

$$\det \begin{pmatrix} 1 & 4 & \square \\ \square & \square & \square \\ -1 & 9 & \square \end{pmatrix} = \det \begin{pmatrix} 1 & 4 \\ -1 & 9 \end{pmatrix} = (9 - (-4)) = 13$$

Since there are $\begin{pmatrix} n \\ k \end{pmatrix}$ ways of selecting k rows from n, and there are $\begin{pmatrix} n \\ k \end{pmatrix}$ ways of selecting k columns from n, there are a total of $\begin{pmatrix} n \\ k \end{pmatrix} \cdot \begin{pmatrix} n \\ k \end{pmatrix}$ minors of order k.

Definition 3.1 *Let A be an $n \times n$ matrix and k an integer with $0 < k \leq n$. A $k \times k$ minor of A is the determinant of the $k \times k$ matrix obtained from A by deleting $n-k$ rows and $n-k$ columns.*

Definition 3.2 *If A is an $n \times n$ matrix, I is a subset of $\{1, ..., n\}$ with k elements, and J is a subset of $\{1, ..., n\}$ with k elements, then let $[A]_{I,J}$ denote the submatrix of A whose row and column indices are respectively taken from I and J.*

Definition 3.3 *If the indices that reference the rows and columns of a matrix A are equal (i.e. $I = J$), then* $\det([A]_{I,J})$ *is called a* principal minor. *If the matrix that corresponds to the principal minor is an upper-left part of the larger matrix (i.e. it is formed by deleting the last $n-k$ rows and the last $n-k$ columns of the matrix A), then the principal minor is called a* leading principal minor. *For an $n \times n$ matrix, there are n leading principal minors. Let $A_{k,k}$ denote a leading principal minor of A of order k.*

Chapter 4

Selecting a Matrix with an Optimal HNF

Hermite Normal Form matrices are used in lattice based cryptography since they are better for answering questions of vector inclusion, without compromising security. Although the techniques for constructing HNF matrices have improved, they do not cater for the need of reducing the key length of the public key cryptosystems nor the digital signature schemes that use them. That is, for selecting private key matrices whose public key HNF matrices can be expressed as a single column of values, as depicted by Definition 4.1.

Definition 4.1 *Let $H \in \mathbb{Z}^{n\times n}$ be the HNF of a matrix $A \in \mathbb{Z}^{n\times n}$. H is* optimal *if for all $i < n, H_{i,i} = 1$.*

For example, the following HNF matrix is in optimal form.

$$H = \begin{bmatrix} 1 & 0 & 0 & 335 \\ 0 & 1 & 0 & 286 \\ 0 & 0 & 1 & 1663 \\ 0 & 0 & 0 & 2873 \end{bmatrix}$$

Notice that, since most diagonal entries of this matrix are equal to 1, it can be expressed by a single column of values, (or a vector), thereby reducing the signature length. Further notice that the determinant of such a matrix is expressed as the product of all the diagonal entries, implying that if a random matrix is carefully chosen to have a prime determinant then, in all likelihood, its HNF will be in optimal form. Note that this is not necessarily the case. That is, there can exist a matrix with a prime determinant that is not in optimal form as the following example illustrates. Suppose one is given the following matrix,

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ where } \mathrm{HNF}(A) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Notice that the determinant of A is prime, yet its HNF is not in optimal form. Clearly, the probability of randomly selecting such a matrix is negligible. Furthermore, if any prime determinant matrix has an HNF that is not optimal, it will be unique with respect to the column where the prime value exists, and hence, one permutation away to being optimal. Therefore, one need only be concerned with selecting a matrix whose determinant is prime. Finding an optimal HNF matrix efficiently, however, is not apparent and can lead to many trial and error attempts before one is selected. Similarly so even if the determinant of the matrix is not prime [RPS11].

One method employed in the paper by Rose, Plantard and Susilo, "Improving BDD Cryptosystems in General Lattices" [RPS11] is to check whether or not the determinants of the randomly chosen matrices are co-prime to a set of small prime factors prior to calculating their HNF. Although the time complexity of calculating the determinant is less than that for calculating the HNF, this can also be an expensive process since, according to Rose *et al.*, only 40% of matrices selected in this manner will result in an optimal HNF.

This chapter proposes yet another method of selecting private key matrices such that their public key HNF matrices are optimal. It takes advantage of the incremental nature of Micciancio and Warinschi's algorithm for calculating Hermite Normal Form matrices. Building a matrix row by column in this manner has revealed that, if the determinant of a principal sub-matrix is prime, then the determinant of the next (or consecutive) principal sub-matrix is also likely to be prime. Since the determinant of a matrix need not be prime in order for it to be optimal, this chapter also looks at an alternative method of selecting optimal HNF matrices which does not entail prime determinants.

4.1 Constructing Matrices with Prime Determinants

Combining the methods employed by Rose *et al.* with the method of searching for large prime numbers (as described in Section 3.2), one could construct a matrix whose determinants are prime or probable prime by multiplying the randomly chosen coefficients by a fixed set of small prime factors, followed by the addition of a unimodular matrix. For example, given the following *randomly* chosen matrix,

$$A = \begin{bmatrix} -25 & 36 & -44 & 20 \\ 15 & -10 & 35 & -12 \\ -52 & 26 & -66 & 72 \\ 16 & -21 & 98 & -30 \end{bmatrix},$$

multiply the following set of prime factors, $\{2, 3, 5, 7\}$, to give,

$$A' = 2 \times 3 \times 5 \times 7 \times A = \begin{bmatrix} -5250 & 7560 & -9240 & 4200 \\ 3150 & -2100 & 7350 & -2520 \\ -10920 & 5460 & -13860 & 15120 \\ 3360 & -4410 & 20580 & -6300 \end{bmatrix}.$$

Next, add this to the following randomly chosen unimodular matrix,

$$U = \begin{bmatrix} 14 & -15 & 14 & 14 \\ 7 & 13 & -8 & -7 \\ -1 & 30 & -21 & 13 \\ -33 & 19 & -20 & 8 \end{bmatrix}$$

to give,

$$U + A' = \begin{bmatrix} -5236 & 7545 & -9226 & 4214 \\ 3157 & -2087 & 7342 & -2527 \\ -10921 & 5490 & -13881 & 15133 \\ 3327 & -4391 & 20560 & -6292 \end{bmatrix}.$$

Notice how the HNF of A is not in optimal form,

$$\mathrm{HNF}(A) = \begin{bmatrix} 1 & 0 & 2 & 265248 \\ 0 & 1 & 1 & 11946 \\ 0 & 0 & 3 & 259964 \\ 0 & 0 & 0 & 303476 \end{bmatrix}.$$

However, the HNF of $U + A'$ is in optimal form,

$$\mathrm{HNF}(U + A') = \begin{bmatrix} 1 & 0 & 0 & 795852267986782 \\ 0 & 1 & 0 & 1364900878811926 \\ 0 & 0 & 1 & 709409421401056 \\ 0 & 0 & 0 & 1784123562021361 \end{bmatrix}.$$

Note that any randomly chosen unimodular matrix may be used. Finally, to determine optimality, one would test whether or not the determinant of the matrix were prime, or probable prime, prior to calculating its HNF.

Algorithm 7 ConstructOptimalHNF

Input: A dimension $n \in \mathbb{N}$, a range of values $r \in \mathbb{N}$, the number of prime factors $k \in \mathbb{N}$, the primes $\{p_1, p_2, \ldots, p_k\}$.

Data: A randomly chosen unimodular matrix U, and a variable Π to store the product of primes.

Output: Two square matrices, $A, H \in \mathbb{Z}^{n \times n}$ such that H is the optimal Hermite Normal Form of A.

```
Π_0 = 1;
for i = 1 to k do
    Π_i ← Π_i × p_i;
end for
U ← unimodular matrix;
repeat
    for i = 1 to n do
        for j = 1 to n do
            A_{i,j} ← Random(−r, r);
        end for
    end for
    A ← U + Π_k × A;
until det(A) is prime (or probable prime);
H ← HNF(A, det(A));
```

The correlation between optimal HNF matrices produced by Algorithm 7, and the determinants being co-prime to a set of small prime numbers, for example, is illustrated in Figure 4.1. The results were obtained by measuring the number of optimal HNF matrices whose determinants were co-prime to a set of small prime

numbers. This was done 1000 times for each dimension between 1 and 300, using coefficients ranging between -500 and 500. Notice that, as the number of prime factors, k, increases, so too does the likelihood of the HNF matrix being optimal.

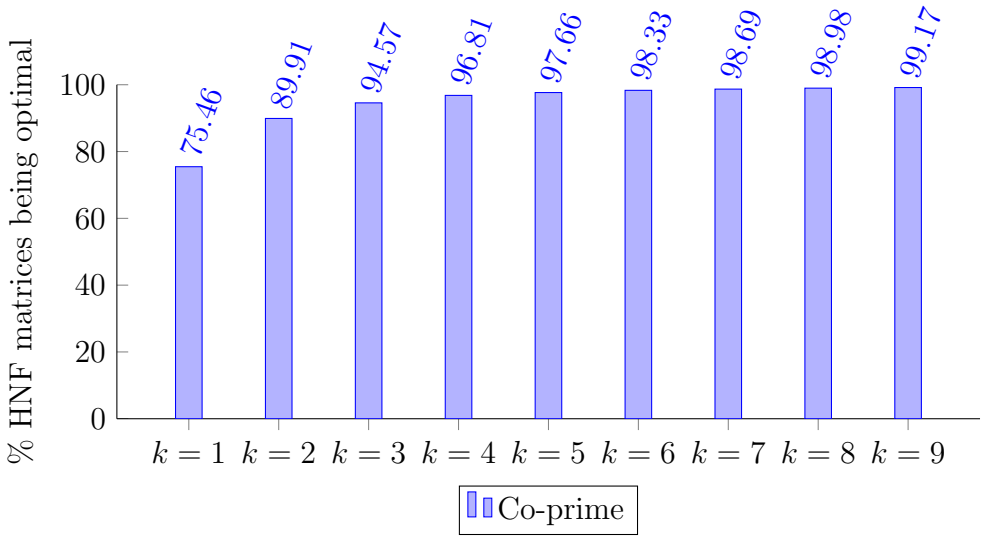

Figure 4.1: Correlation between optimal HNF matrices produced by Algorithm 7, and the determinants being co-prime to a set of small prime numbers.

In terms of efficiency, this method was compared to the method proposed by Rose *et al.* As such, the test for optimality was to ensure that the determinant of the constructed matrix was co-prime to a set of small prime values. The results are illustrated in Figure 4.2. Notice that there is an asymptotic improvement in performance as the dimension of the matrix increases. This is because the time of finding a determinant of a randomly selected matrix that is co-prime to a set of small prime values is reduced.

This method works since, like Rose *et al.* method, it forces the determinant of a matrix to be co-prime to a set of small prime values, thereby increasing its likelihood of being prime. This method is also practical since it leads more directly to a probable solution than searching for a subset of possible solutions. This is ideal for fully homomorphic encryption algorithms such as those noted by Gentry [GPV08] who uses optimal HNF matrices with dimensions as large as 2048×2048 in size. Unfortunately, the method does not guarantee a prime determinant which is required for fully homomorphic encryption algorithms such as those noted by Smart [SV10].

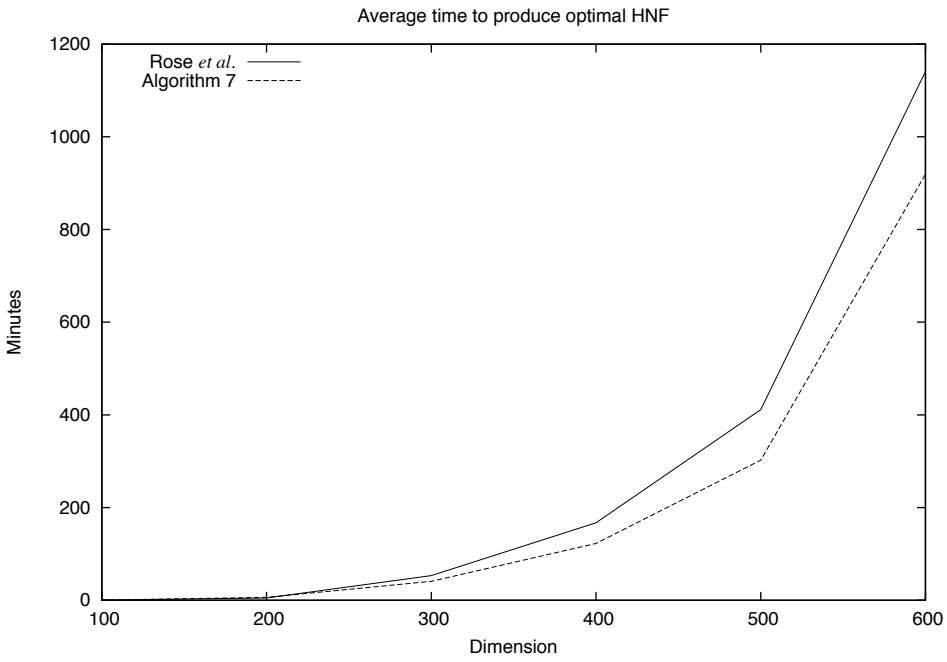

Figure 4.2: Comparison between Rose *et al.* and proposed method of computing an optimal HNF matrix, in terms of dimension versus time.

4.2 Selecting Matrices with Prime Determinants

This section explores different methods of generating non-singular square matrices whose HNFs are in optimal form. It explores ways of selecting matrices (a.k.a. private keys) such that their determinants are prime (or probable prime). As noted earlier, if the determinant of the matrix is prime, there is a more than likely chance that its HNF will be in optimal form.

This section begins by describing the traditional method of selecting matrices such that their HNF's are in optimal form (Section 4.2.1). This is followed by the proposed method of selecting such matrices row by column using Micciancio and Warinschi's algorithm to calculate Hermite Normal Forms (Section 4.2.2). The idea behind this method is that by forcing all leading principal minors to being prime during matrix selection will force the determinant of the target matrix to be prime.

4.2.1 Traditional Method of Selecting Matrices

In its most simplest form, the traditional (trial and error) method of selecting and calculating an optimal HNF matrix is to randomly select an $n \times n$ matrix, calculate its HNF, and then determine its optimality. If the HNF matrix is not optimal, the

randomly chosen matrix is discarded, and the selection process started again. This is illustrated in Algorithm 8.

Algorithm 8 TraditionalOptimalHNF

Input: A dimension n, and a range r.

Output: Two square matrices, $A, H \in \mathbb{Z}^{n\times n}$ such that H is the optimal Hermite Normal Form of A.

```
repeat
  for i ← 1 to n do
    for j ← 1 to n do
      A_{i,j} ← Random Integer between −r and r;
    end for
  end for
  H = HNF(A);
until det(H) is optimised;
```

Note that the test for optimality can be any one of the methods noted in section 3.1, although alternative tests that don't involve prime determinants may be used and are discussed in Section 4.3.

Assuming that the time for randomly choosing coefficients of a matrix A is negligible, the time for selecting and computing an optimal HNF matrix, H, is given by the following formula

$$T_1 = a_1 \times \left(\sum_{i=1}^{n} (T_{\text{AddRow}}(i) + T_{\text{AddColumn}}(i)) + T_{\text{Prime}_k}(n) \right)$$

where T_{AddRow} and $T_{\text{AddColumn}}$ denote the times for executing AddRow and AddColumn, respectively, T_{Prime_k} denotes the time for testing a prime determinant using the Miller-Rabin method with k witnesses, and a_1 denotes the average number of steps for selecting a matrix whose HNF is in optimal form.

The coefficient a_1 is calculated by taking the average of probability, p_1, of successfully finding a matrix whose HNF is in optimal form. This is given by the following formula

$$\begin{aligned} a_1 &= 1 \times p_1 + 2 \times (1-p_1)p_1 + 3 \times (1-p_1)^2 p_1 + \cdots \\ &= \sum_{i=1}^{\infty} i(1-p_1)^{i-1} p_1 = \frac{p_1}{p_1^2} = p_1^{-1} \end{aligned} \tag{4.1}$$

4.2.2 Selecting Matrices whose Leading Principal Minors are Prime

A viable method of selecting a matrix, A, whose HNF is optimal is to construct it row by column such that the leading principal minor of the matrix is prime. For example, at the first iteration, the sub matrix $A_{\{1\},\{1\}}$ of A (that is, the first leading principal minor) should be chosen to have a single prime number.

$$A_{\{1\},\{1\}} = [7], \quad \text{HNF}(A_{\{1\},\{1\}}) = [7]$$

At the second iteration, three numbers should be chosen to form the outer row and column of $A_{\{1\},\{1\}}$ such that the determinant of the resulting 2×2 matrix, $A_{\{1,2\},\{1,2\}}$, (that is, the second leading principal minor) is prime.

$$A_{\{1,2\},\{1,2\}} = \begin{bmatrix} 7 & -5 \\ -3 & 8 \end{bmatrix}, \quad \text{HNF}(A_{\{1,2\},\{1,2\}}) = \begin{bmatrix} 1 & 11 \\ 0 & 41 \end{bmatrix}.$$

At the third iteration, five numbers should be chosen to form the outer row and column of $A_{\{1,2\},\{1,2\}}$ such that the determinant of the resulting 3×3 matrix, $A_{\{1,2,3\},\{1,2,3\}}$, (that is, the third leading principal minor) is prime.

$$A_{\{1,2,3\},\{1,2,3\}} = \begin{bmatrix} 7 & -5 & 2 \\ -3 & 8 & -9 \\ 1 & -4 & 6 \end{bmatrix}, \quad \text{HNF}(A_{\{1,2,3\},\{1,2,3\}}) = \begin{bmatrix} 1 & 0 & 44 \\ 0 & 1 & 33 \\ 0 & 0 & 47 \end{bmatrix}.$$

And, so forth until the required matrix dimension of A is reached. This is illustrated in Algorithm 9.

Algorithm 9 CreatePrimeHNF

Input: A dimension n, and a range r.

Data: A vector $\hat{a} \in \mathbb{Z}^{i-1}$ to store the column that will be added; a vector $\hat{b} \in \mathbb{Z}^{i}$ to store the row that will be added; and a vector $\hat{x} \in \mathbb{Z}^{i-1}$ to store the column that has been added.

Output: Two square matrices, $A, H \in \mathbb{Z}^{n \times n}$ such that H is the optimal Hermite Normal Form of A.

$H_{1,1} \leftarrow A_{1,1} \leftarrow$ GenPrime();
for $i = 2$ **to** n **do**
  repeat
    for $j = 1$ **to** $i - 1$ **do**
      $\hat{a}_j \leftarrow A_{i,j} \leftarrow$ Random Integer between $-r$ and r;
    end for
    $\hat{x}^T(i-1) \leftarrow$ AddColumn$(A(i-1), H(i-1), \hat{a}^T(i-1))$;
    for $j = 1$ **to** i **do**
      $\hat{b}_j \leftarrow A_{i,j} \leftarrow$ Random Integer between $-r$ and r;
    end for
    $H_{i,i} \leftarrow$ AddRow$\left(\begin{bmatrix} H(i-1) & \hat{x}^T(i-1) \end{bmatrix}, \begin{pmatrix} \hat{b}_{i,1}, \ldots, \hat{b}_{i,i} \end{pmatrix}\right)$;
  until $\det(H_{i,i})$ is prime;
end for

Note that the HNF of each (leading principal) sub-matrix is calculated at each iteration. If the determinant of the resulting HNF matrix is not prime, the newly added row and column are simply discarded, and numbers for those chosen again, as opposed to discarding the whole matrix. Further note that the condition that tests for a prime determinant could easily be replaced by a condition that tests for a probable prime determinant.

Again, the test for optimality can be any of the methods noted in section 3.1. Assuming that the time for randomly choosing coefficients of a row/column are negligible, the time for selecting and computing an optimal HNF matrix, using this method, is given by the following formula

$$T_2 = a_2 \times \left(\sum_{i=2}^{n} T_{\text{AddRow}}(i) + T_{\text{AddColumn}}(i) + T_{\text{Prime}_k}(i) \right)$$

where T_{AddRow} and $T_{\text{AddColumn}}$ denote the times for executing AddRow and AddColumn, respectively, T_{Prime_k} denotes the time for testing a prime leading principal

minor using the Miller-Rabin method with k witnesses, and a_2 denotes the average number of steps for selecting a matrix whose HNF is in optimal form.

Similarly, a_2 is calculated by taking the average of probability, p_2, of successfully finding a matrix whose HNF is in optimal form. This is given by the following formula

$$\begin{aligned} a_2 &= 1 \times p_2 + 2 \times (1-p_2)p_2 + 3 \times (1-p_2)^2 p_2 + \cdots \\ &= \sum_{i=1}^{\infty} i(1-p_2)^{i-1} p_2 = \frac{p_2}{p_2^2} = p_2^{-1} \end{aligned} \tag{4.2}$$

4.2.3 Complexity Analysis between Traditional and Proposed Methods

To analyse the complexity between the traditional and proposed methods of finding matrices whose Hermite Normal Forms are in optimal form, one needs to compare the average number of steps in terms of the probability of success. That is, a_1 and a_2 in terms of p_1 and p_2. Note that, if T_{PRIME_k} is negligible compared to T_{ADDROW} and $T_{\text{ADDCOLUMN}}$, then $T_2 < T_1$ if $a_2 < a_1$.

It is clear that if the average number of steps of using the proposed method is superior to that of using the traditional approach, then $a_2 < a_1$. By equations (4.1) and (4.2), this implies that $p_2^{-1} < p_1^{-1}$, which further implies that if $p_1 < p_2$, then the proposed method is asymptotically faster than the traditional approach. This is demonstrated in section 4.2.4, which analyses the probability of success between the two methods of finding matrices whose Hermite Normal Forms are in optimal form.

4.2.4 Empirical Analysis between Traditional and Proposed Methods

It was observed that selecting a row/column pair whose leading principal minor was prime increased the likelihood of the next leading principal minor being prime. This is also true for leading principal minors that were probable prime. To verify this behaviour, though, a number of tests were conducted independent of the above algorithm. The purpose of these tests was to also confirm whether this behaviour was limited to leading principal minors, or any minor of randomly chosen matrices.

Testing for Prime Determinants

To demonstrate that the primality of leading principal minors influence the primality of consecutive leading principal minors, the determinants of randomly chosen matrices, along with their leading principal minors, were tested for primality. As such, the test was constructed to count the number of consecutive prime minors, as well as consecutive non-prime minors. If the leading principal minors of two consecutive sub-matrices were prime, (that is, if the $\det(A_{i,i})$ were prime, and the $\det(A_{i+1,i+1})$ were also prime), this would be added to a counter. Similarly, if the leading principal minors of two consecutive sub-matrices were not prime, this result would be added to yet another counter. Counters were also maintained for the cases where the ith leading principal minors were prime, and the $(i+1)$th leading principal minors were not, and vice-versa.

The dimension of the matrices concerned were 200×200, the leading principal minors of which were all tested for primality, except for the first and last leading principal minors since there was nothing to compare them against. The range of values used for the coefficients was between -200 and 200; the dimension of the matrices concerned. Finally, the experiments were repeated 1000 times for each leading principal minor tested, and the results averaged. Table 4.1 illustrates the results of this test.

	% Prime Distribution				Probability	
	P/P	P/NP	NP/P	NP/NP	p_1	p_2
Stricter Prime	0.0055	0.2075	0.2075	99.0795	0.00213	0.02582
Non-Deterministic	3.55	10.75	10.82	74.88	0.14370	0.24825

Table 4.1: Distribution of leading principal minors being prime compared to the percentage of matrices having a prime determinant, using both non-deterministic and stricter prime methods of determining primality (see explanation below).

Initially, the Rabin-Miller method with one witness was chosen to perform each test, although this could have been replaced by trial divisions of a set of small prime numbers. Unless otherwise specified, this is what is meant by employing a *non-deterministic* or *probable prime* method. For more accurate results, the number of witnesses would be increased to 10. Although more accurate, this is still considered non-deterministic. For more definitive results, **magma**® which provides a more rigorous method for testing prime numbers was used. It accurately determines the primality of any number n, unless $n > 25 \times 10^9$ or the optional parameter '*Proof*'

is set to false, in which case the result indicates that n is probable prime using the Rabin-Miller method with 20 witnesses. Because most of the determinants that are calculated are greater than this specified bound of accuracy, most of the positive results returned by **magma**® will be probable prime at best. Unless otherwise specified, this is what is meant by employing a *stricter prime* method.

The reader will note that, although the likelihood of two consecutive leading principal minors being prime is low, and the likelihood of two consecutive leading principal minors not being prime is high, the probability of success using the proposed method is greater than that of using the traditional method (especially when a stricter prime approach of identifying primary determinants is utilised). What this implies is that the average number of steps required to successfully select a matrix whose HNF is in optimal form is less using the proposed method than it is using the traditional method, as illustrated in Table 4.2.

	a_1	a_2	Ratio
Stricter Prime	469.4836	38.7273	12.1228
Non-Deterministic	6.9589	4.0282	1.7276

Table 4.2: Average number of steps of successfully selecting a matrix whose HNF is in optimal form using both stricter prime and non-deterministic methods of determining primality.

What is even more interesting is the fact that this observation occurs for all consecutive leading principal minors. That is, the likelihood of this occurring for the $(i-1)$th and ith leading principal minors is the same as it occurring for the $(n-1)$th and nth leading principal minors, as illustrated in Figure 4.3 (not counting the first and last measurements, since there is nothing to compare them against).

These measurements were taken by counting the number of leading principal minors that were prime, as well as those that were not prime. In particular, a counter was maintained for two consecutive leading principal minors being prime, as well as a counter for two consecutive principal minors not being prime. Similarly, counters were maintained for two consecutive leading principal minors that complemented each other in terms of primality (that is, prime to non-prime, and non-prime to prime). Notice how the number of consecutive leading principal minors that are prime remains constant, as do all those that are non-prime, as the dimension of the matrix (leading principal minor) increases. Also notice how the number of consecutive leading principal minors that complement each other in terms of primality also

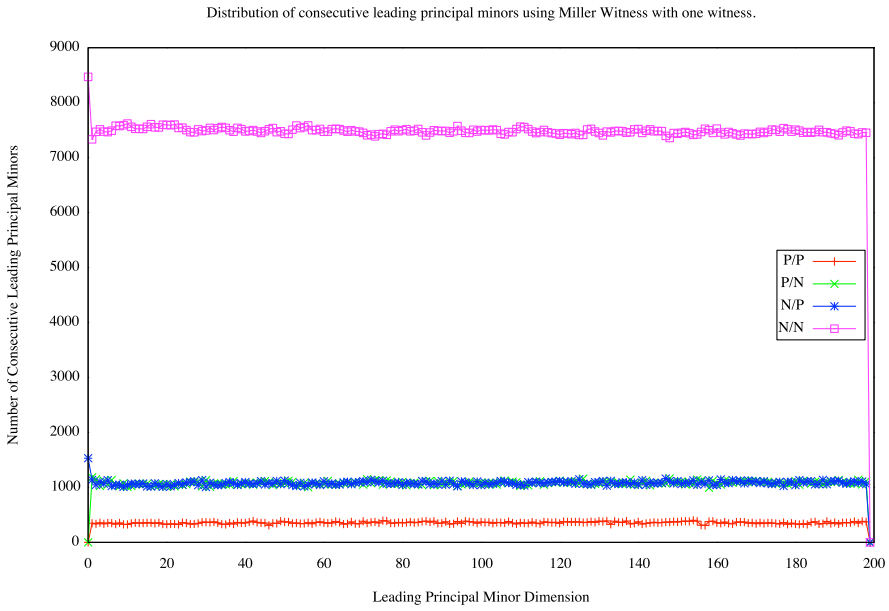

Figure 4.3: Distribution of Consecutive Leading Principal Minors being Prime using Rabin-Miller method with one witness.

remains constant.

This observation warranted further investigation, since using the Rabin-Miller method with one witness (a.k.a. non-deterministic methods) could bias the results. As such, further tests were conducted by increasing the number of Miller witnesses to 10. Sure enough, this provided a more accurate bound on the number of leading principal minors that were prime, that is consistent with the bound of prime determinants on matrices noted by Maples [Map10].

As the Miller-Rabin test is probabilistic, there is a possibility that some minors were mistaken to be prime, but in fact were not. Because of this, a stricter prime method for predicting primality was employed. That is, the rigorous methods provided by **magma**®. This had a negligible difference in results, as illustrated in Table 4.3. Noticeably, there is still a pattern that holds true, although now there are fewer consecutive leading principal minors that are both prime.

Nevertheless, it was observed that using a stricter prime approach of determining prime leading principal minors saw a significant improvement in performance. Specifically, the average number of steps for selecting a matrix whose HNF is in optimal form was reduced by a factor of 12 as illustrated in Table 4.4. This is unlike the non-deterministic approach which only saw a reduction by a factor of 1.7. This is because there is a notable improvement in applying non-deterministic methods to

	% Prime Distribution				Probability	
	P/P	P/NP	NP/P	NP/NP	p_1	p_2
Non-Deterministic	3.5915	10.7580	10.7580	74.3925	0.1437	0.2483
Rabin-Miller (10 witnesses)	0.0055	0.2075	0.2075	99.0795	0.00213	0.0258
Stricter Prime	0.0055	0.2075	0.2075	99.0795	0.00213	0.0258

Table 4.3: Distribution of prime leading principal minors using non-deterministic, Rabin-Miller with 10 witnesses, and stricter prime methods of determining primality.

traditional trial and error techniques as illustrated in Table 4.3.

	a_1	a_2	Ratio
Non-Deterministic	6.9589	4.0282	1.7276
Rabin-Miller (10 witnesses)	469.4836	38.7273	12.1228
Stricter Prime	469.4836	38.7273	12.1228

Table 4.4: Average number of steps of successfully selecting a matrix whose HNF is in optimal form using non-deterministic, Rabin-Miller with 10 witnesses, and stricter prime methods of determining primality.

To determine whether or not any minors other than the leading principal minors affects or influences the primality of the determinant, the following test was conducted. The determinants of randomly chosen matrices, along with a subset of their minors, were tested for primality. As such, the test was constructed to count the number of prime minors, as well as non-prime minors, for determinants that were both prime and not prime. Since it is not practical to test all the minors of any matrix, a subset of minors was chosen to perform this test. As such, matrices of dimension 50×50 were created, with coefficients ranging from -50 to 50. Their minors were selected by successively deleting one row and one column, and then calculating the determinant of the associated 49×49 matrices. For example, on the first loop, the first row and the first column of the 50×50 matrix would be deleted, and the determinant calculated; on the second loop, the first row and second column of the same 50×50 matrix would be deleted, and the determinant calculated; and so forth. In total, 10,000 repetitions of each minor selected was tested for primality, and the results averaged. Sure enough, the results revealed a correlation between prime (or probable prime) minors and the determinants of the matrices, as depicted in Table 4.5.

Notice how these results closely resemble those of p_2 of Table 4.3. That is, the

	Determinant		
	Prime	Non-Prime	Ratio
Non-Deterministic	0.108647	0.044498	2.44162
Rabin-Miller (10 witnesses)	0.050000	0.001803	27.73160
Stricter Prime	0.050000	0.001803	27.73160

Table 4.5: Percentage of prime minors resulting in a prime determinant versus the percentage of prime minors resulting in a non-prime determinant using trial divisions, Rabin-Miller with 10 witnesses, and stricter prime methods of determining primality.

probability of finding a prime determinant using arbitrary minors closely resembles the probability of finding a prime determinant using leading principal minors. Further notice that the probability of finding a prime determinant increases using non-deterministic or probable prime methods as opposed to stricter prime methods.

Interestingly, the results obtained also indicated that if there were no prime minors, then the determinant of matrix would not be prime either. This is not to say that prime determinants of matrices could not exist in their own right without any of their minors being prime. One such example is the following matrix,

$$\begin{bmatrix} 10 & 1 \\ 93 & 10 \end{bmatrix}.$$

Note that the determinant of the matrix is 7, yet all of its minors (in this case, the coefficients) are non-prime. This, however, is not a good example as it relies of the fact that 1 is not a prime number by definition. The subset of results collected thus far have failed to find combinations of non-prime minors whose determinants are prime. The converse to this, however, is not true since there did exist some minors that were prime, but the determinant of the matrix wasn't.

Thus far, the range of randomly selected matrix coefficients used to test this property lies between $-r$ and r, where $r = n$, the dimension of the target matrix. To demonstrate that this property behaves the same regardless of the range, the above tests were repeated using the ranges $\{-1, 0, 1\}$ and $\{-2^{32} + 1, \ldots, 2^{32} - 1\}$, respectively. Furthermore, the Rabin-Miller method with one witness was employed to demonstrate consistency between this and previous tests. Suffice it to say that there was only a slight improvement in behaviour, as is illustrated in Table 4.6.

	% Prime Distribution				Probability	
	P/P	P/NP	NP/P	NP/NP	p_1	p_2
$\{-1, 0, 1\}$	3.313	10.175	10.175	75.837	0.1356	0.2456
$\{-n, ..., n\}$	3.55	10.75	10.82	74.88	0.1437	0.2483
$\{-2^{32}+1, \ldots, 2^{32}-1\}$	3.659	10.8095	10.8095	74.222	0.1454	0.2529

Table 4.6: Distribution of prime determinants using different ranges, and the Rabin-Miller method with one witness to determine primality of leading principal minors. Note the consistency between this table and Table 4.1.

Timing

A comparison was made between adding a single row/column pair using Algorithm 9 and testing for a prime determinant. The aim was to see how expensive testing for a prime determinant is compared to adding a single row/column pair, which includes calculating its determinant. As such, both tests were repeated 1000 times for each dimension, and the timings of each test recorded. For prime testing, the **ProbPrime**(**const ZZ&** n, **long** *NumTrials*) function of the NTL class of functions for large integers (i.e. **ZZ**) was employed, which performs up to *NumTrials* Miller witness tests after some trial divisions. A parameter of $NumTrials = 0$ will therefore only check that n (i.e. the determinant) is co-prime to a set of small prime numbers. The cumulative timing results illustrated in Figure 4.4 were obtained on a quad core Intel(R) Xeon(R) CPU X3360 running at 2.83GHz under the GNU/Linux operating system with 8GB RAM.

It is clear that checking for a prime determinant is negligible as the dimension of the matrix grows. Even checking that the determinant is co-prime to a set of small prime numbers alone is negligible. For example, at dimension 250, the effort of adding a single row/column pair, and computing the corresponding determinant of the matrix, is far more expensive than testing for a prime determinant. Increasing the number of witnesses and/or increasing the number of prime factors would, therefore, have little impact on performance.

Adding a single row/column pair consistent with Algorithm 9, however, is an expensive operation, especially as the dimension of the matrix grows. Limiting the number of trial and error attempts in finding a matrix whose HNF is optimal would therefore benefit the selection process.

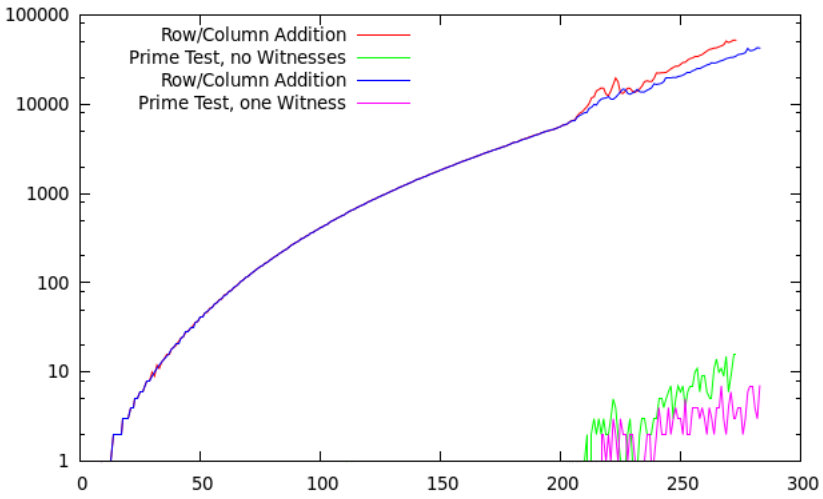

Figure 4.4: Cumulative time of adding row/column pair compared to prime testing of determinants using Miller Witness. Note that the time for each was recorded 1000 times for each dimension of the matrix constructed.

4.3 Selecting Matrices whose HNF's are Optimal

As noted earlier, a matrix need not have a prime determinant in order for its HNF to be in optimal form. Hence, rather than checking for primality of the determinant at each iteration of the algorithm, one could check for optimality of the HNF matrix. Like checking for primality of the leading principal minor influences the primality of the consecutive leading principal minors, so too does checking for optimality; in fact, much better than randomly selecting a matrix such that its HNF is in optimal form.

To demonstrate this, the HNF of a group of randomly selected matrices were tested for optimality. They were built row by column using the above HNF algorithm. Like before, the number of consecutive optimal matrices were counted, as well as the number of consecutive non-optimal matrices. Similarly, the number of consecutive matrices that complemented each other in terms of optimality were also counted (that is, if $H_{i,i}$ were optimal and $H_{i+1,i+1}$ were not, and vice-versa). The results are illustrated in Table 4.7.

It is apparent that the chances of selecting a random matrix whose HNF is in optimal form is far greater using this method than testing for primality. Although the chances of selecting a random matrix whose HNF is non-optimal are greater, like before, the probability of success using the proposed method is greater than that of using the traditional method. What this implies is that the average number of

	% Optimal Distribution				Probability	
	O/O	O/NO	NO/O	NO/NO	p_1	p_2
Optimal	26.24	17.14	17.95	38.67	0.4419	0.6049

Table 4.7: Distribution of consecutive optimal HNF matrices versus non-optimal HNF matrices using a method other than prime determinants for determining 'optimality'.

steps required to successfully select a matrix whose HNF is in optimal form is less using the proposed method than it is using the traditional method, as illustrated in Table 4.8. Note, however, that the result is not as significant as that of testing for a prime leading principal minor using stricter prime methods. This is because there is a notable improvement in applying optimal testing methods to traditional trial and error techniques as illustrated in Table 4.7.

	a_1	a_2	Ratio
Optimal	2.2630	1.6532	1.3688

Table 4.8: Average number of steps of successfully selecting a matrix whose HNF is in optimal form using a method other than prime determinants for determining 'optimality'.

Also, like before, this phenomenon occurs for all consecutive leading principal matrices, as is illustrated in Figure 4.5 (again, not counting the first or last measurements as there is nothing to compare them to).

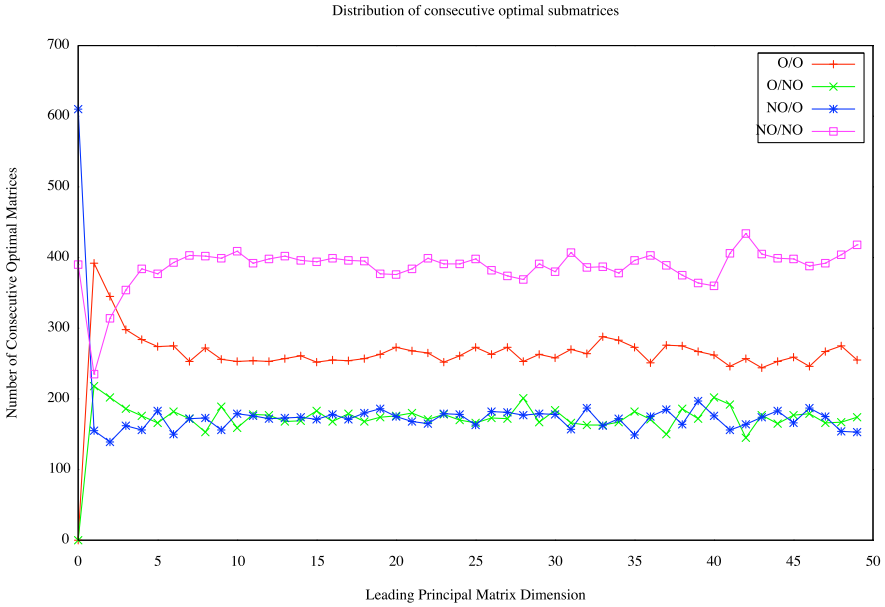

Figure 4.5: Distribution of Consecutive Optimal Submatrices.

Chapter 5

Conclusion and Future Work

Hermite Normal Form matrices are a standard form of integer matrices used in applications such as integer programming, loop optimisation, and for solving linear Diophantine equations. They are also used for reducing the key length and ciphertext size of lattice-based cryptosystems and trapdoor functions of the kind proposed by Goldreich, Goldwasser and Halevi, without decreasing their security. Although the techniques for constructing Hermite Normal Form matrices have improved, a method of selecting a random matrix whose Hermite Normal Form is in optimal form has not. It remains a trial and error process at best, with a 40% chance of success.

The main objective of this thesis was to explore ways of reducing the trial and error associated with this process. In particular, it studied a stochastic observation that results in the determinants of randomly selected matrices being prime – a further observation being that most Hermite Normal Forms of such matrices can be expressed as a single column of values. Also studied was a sieve method that also aided in reducing the time of the selection process. The determinants of the resulting matrices using this method, however, were not guaranteed to be strictly prime. Since the determinant of randomly selected matrices need not be prime in order for their Hermite Normal Forms to be in optimal form, other methods of selecting matrices were further studied with varying degrees of success.

5.1 Contribution

Testing revealed that if a leading principal minor of a matrix were prime, it increased the likelihood of the next (or consecutive) leading principal minor being prime, as illustrated in Tables 4.1, 4.2 and 4.3. Similarly so, if the Hermite Normal Form of a leading principal sub-matrix were chosen to be optimal, as illustrated in Table 4.7.

As such, tests were conducted to verify the phenomenon, with varying degrees of success. Test results revealed that if a stricter prime approach of identifying prime minors was employed (that is, increasing the number of Rabin-Miller witnesses to 20), it reduced the average number of steps of selecting a matrix whose determinant is prime by a factor of 12. This was also true for increasing the number of Rabin-Miller witnesses to 10.

Taking a non-deterministic (or probable prime) approach of determining prime minors showed a better improvement in selecting optimal matrices, as did traditional trial and error techniques employing non-determistic (or probable prime) methods of identifying prime determinants. Because of this, the average number of steps of selecting a matrix whose determinant was probable prime using the proposed method was reduced only by a factor of 1.7 when compared to traditional trial and error techniques. This was also true for selecting leading principal sub-matrices whose Hermite Normal Forms were optimal, which showed an even better improvement in performance, but only a reduction in the average number of steps by a factor of 1.3.

Testing for a prime determinant, on the other hand, had negligible impact on the algorithm, meaning that one can increase the number of Rabin-Miller witnesses with minimal impact on the overall performance. This is ideal for fully homomorphic encryption algorithms noted by Smart [SV10], who relies on the determinant of large HNF matrices being strictly prime. It further improves the security of the cryptosystem and/or trap-door function.

Also studied was a sieve method that also reduced the time of selecting a matrix whose Hermite Normal Form is in optimal form. This method employed probable prime techniques of filtering prime determinants by ensuring they were co-prime to a set of small prime numbers. The results revealed that increasing the number of prime factors saw a remarkable correlation between optimal HNF matrices and (probable) prime determinants. So much so that it led more directly to a probable solution rather than searching for a possible subset of solutions. This is ideal for fully homomorphic encryption algorithms such as those noted by Gentry *et al.* [GPV08], who use optimal HNF matrices with dimensions as large as 2048×2048 in size.

5.2 Future Work

With the subset of data collected, it was observed that if there were no prime minors, the determinants of the matrices were not prime, either. A more exhaustive

search of all non-zero minors, however, may prove otherwise. Nevertheless, the equivalence partition of minors selected for testing is believed to be an adequate representation of all the non-zero minors, including the leading principal minors, that form the matrix. This is due to the uniformity and non-bias of the minors that were selected for testing. As such, any observation made could be stated with a certain degree of confidence. Even if this observation is refuted, it does not refute the fact that a correlation between prime minors and prime determinants exists. Therefore, one possible direction for further research is to establish a theoretical proof of the phenomena observed in this thesis.

The decision to use Micciancio and Warinschi's space saving algorithm is due to its incremental nature of calculating HNF matrices row by column. It lends itself more easily to determining optimality of HNF matrices constructed in this manner. Since the sieve method of selecting matrices does not rely on this incremental nature, it would be interesting to note whether or not using Micciancio and Warinschi's heuristic algorithm, or any other HNF algorithm for that matter, would make any difference in performance. Also, as the cost of increasing the number of Rabin-Miller witnesses is negligible in comparison to calculating HNF matrices, it would further be interesting to note the differences of combining any of the techniques discussed in this thesis with the sieve method. Another possible area of research, for example, is to determine whether or not selecting only prime coefficients will impact the performance and the security of the public key that is constructed using the sieve method.

With respect to Algorithm 9, the process of discarding a new row and column if the minor of the leading principal sub-matrix is not prime could be sped up by just modifying a few coefficients, as opposed to selecting an entirely new row and column, and also using the fact that the determinant of $H_{i,i}$ can be calculated using the previously calculated leading principal minors. Furthermore, it could even be possible to fix the column, for example, and then introduce a series of variables for the row, compute the determinant, and then substitute different values in the variables till a prime determinant is found. This prevents recomputing the determinant from scratch every time. The only drawback is that it will more than likely be susceptible to cryptographic attacks due to the nature of the selection process, as opposed to randomly selecting an entirely new row and column.

In terms of lattice based cryptosystems, it remains to see whether or not the suggested methods produce public keys that are resistant to cryptographic attacks.

It is anticipated that the matrices constructed in this manner are just as resilient as those constructed using traditional methods. For security reasons, however, it is recommended that the number of Rabin-Miller witnesses be increased. For other applications that have no security requirement, relaxing the stricter rules of primality may be adequate.

Bibliography

[Bab86] László Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, March 1986.

[Bat11] Batut, C. and Belabas, K. and Bernardi, D. and Cohen, H. and Olivier, M. *PARI/GP, version* `2.5.0`. The PARI Group, Bordeaux, 2011. available from `http://pari.math.u-bordeaux.fr/`.

[BCFS10] W. Bosma, J. J. Cannon, C. Fieker, and A. (eds.) Steel. *Handbook of Magma functions*. Computational Algebra Group, School of Mathematics and Statistics, University of Sydney, 2 edition, 2010. 5017 pages.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[Bra71] Gordon H. Bradley. Algorithms for hermite and smith normal matrices and linear diophantine equations. *Mathematics of Computation*, 25(116):897–907, October 1971.

[Bru16] Viggo Brun. Omfordelingen av primtallene i forskjellige talklasser. en vre begrnsning. *Nyt Tiddsskr. f. Math.*, 27(B):45–58, 1916.

[Bru19] Viggo Brun. Le crible d'Eratostne et le thorme de Goldbach. *C. R. Acad. Sci. Paris*, 168:544–546, 1919.

[Bru22] Viggo Brun. Das siev des eratosthenes. *5. Skand. Mat. Knogr., Helsingfors*, pages 197–203, 1922.

[CC82] Tsu-Wu J. Chou and George E. Collins. Algorithms for the solution of systems of linear diophantine equations. *SIAM Journal on Computing*, 11(4):687–708, 1982.

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer, Heidelberg, 1993.

[CW87] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 1–6, New York, NY, USA, 1987. ACM.

[Dir37] L. Dirichlet. Beweis des satzes, dass jede unbegrenzte arithmetische progression, deren erstes glied und differenz ganze zahlen ohne gemeinschaftlichen factor sing, unendlich viele primzahlen erhält, 1837.

[Dix82] John D. Dixon. Exact solution of linear equations using p-adic expansions. *Numerische Mathematik*, 40:137–141, 1982.

[DKT87] 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, February 1987.

[dlVP96] Charles-Jean de la Vallée Poussin. Recherches analytiques sur la théorie des nombres; Première partie: La fonction $\zeta(s)$ de riemann et les nombres premiers en général. *Annales de la Soc. scientifiques de Bruxelles*, 20(II):183—256, 1896.

[Dwo98] Cynthia Dwork. Lattices and Their Application to Cryptography. Lecture notes, Standford University, IBM Almaden Research Centre, June 13, 1998.

[EW05] Graham Everest and Thomas Ward. *An Introduction to Number Theory*. Graduate Texts in Mathematics. Springer, London, UK, 2005.

[FG89] John Friedlander and Andrew Granville. Limitations to the equidistribution of primes I. *The Annals of Mathematics*, 129(2):pp. 363–382, 1989.

[Fru77] M. Frumkin. Polynomial time algorithms in the theory of linear diophantine equations. In Marek Karpinski, editor, *Fundamentals of Computation Theory*, volume 56 of *Lecture Notes in Computer Science*, pages 386–392. Springer Berlin / Heidelberg, 1977.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In B.S. Kaliski Jr., editor, *Advances in Cryptography - CRYPTO 1997*, volume 1294/1997 of *Lecture Notes in Computer Science*, pages 112–131. Springer Berlin / Heidelberg, 1997.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC '08: Proceedings of the 40th annual ACM symposium on theory for computing*, pages 197–206, New York, NY, USA, May 17–20, 2008. ACM.

[Had96] Jacques Hadamard. Sur la distribution des zéros de la fonction $\zeta(s)$ et ses conséquences arithmétiques. *Bull. Soc. Math. France*, 14:199–220, 1896.

[HHGP+03] Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H. Silverman, and William Whyte. *NTRUSign: Digital Signatures using the NTRU Lattice*, pages 122–140. Number 2612 in Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003.

[HM89] James L. Hafner and Kevin S. McCurley. A rigorous subexponential algorithm for computation of class groups. *Journal of the American Mathematical Society*, 2(4):pp. 837–850, 1989.

[HM90] James L. Hafner and Kevin S. McCurley. Asymptotically fast triangulation of matrices over rings. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, SODA '90, pages 194–200, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[HM91] James L. Hafner and Kevin S. McCurley. Asymptotically fast triangularization of matrices over rings. *SIAM J. Comput.*, 20:1068–1083, December 1991.

[HPS08] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *An Introduction to Mathematical Cryptography*. Undergraduate Texts in Mathematiccs. Springer, 2008.

[HR90] Ming S. Hung and Walter O. Rom. An application of the hermite normal form in integer programming. *Linear Algebra and its Applications*, 140:163–179, 1990.

[Ili89] Costas 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 on Computing*, 18:658–669, 1989.

[KB79] Ravindran Kannan and Achim Bachem. Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix. *SIAM Journal of Computing*, 8(4):499–507, 1979.

[Kow08] Emmanuel Kowalski. *The Large Sieve and its Applications: Arithmetic Geometry, Random Walks and Discrete Groups*, volume 175 of *Cambridge Tracts in Mathematics*. Cambridge University Press, 2008.

[Lit14] J. E. Littlewood. Sur la distribution des nombres premiers. In *"Collected Papers"*, volume 2, pages 263–266. C. R. Acad. Sci. Paris, 1914.

[Map10] Kenneth Maples. A Brun-Titchmarsh inequality for discrete random matrices. Submitted, November 2010.

[MG02] Daniele Micciancio and Shafi Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, March 2002.

[Mic01] Daniele Micciancio. *Improving Lattice Based Cryptosystems using the Hermite Normal Form*, volume 2146 of *Lecture Notes in Computer Science*, pages 126–145. Springer Berlin / Heidelberg, 2001.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

[Mot75] Yoichi Motohashi. On some improvements of the Brun-Titchmarsh theorem. *J. Math. Soc. Japan*, 27:444–453, 1975.

[MS99] Thom Mulders and Arne Storjohann. Diophantine linear system solving. In *Proceedings of the 1999 international symposium on Symbolic and algebraic computation*, ISSAC '99, pages 181–188, New York, NY, USA, 1999. ACM.

[MW01] Daniele Micciancio and Bogdan Warinschi. A linear space algorithm for computing the hermite normal form. In *ISSAC '01: Proceedings of the 2001 international symposium on Symbolic and algebraic computation*, pages 231–236, New York, NY, USA, 2001. ACM.

[Ngu99] Phong Q. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem from Crypto 1997. In Michael Wiener, editor, *Advances in Cryptography - CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 288–304. Springer Berlin / Heidelberg, 1999.

[NR06] Phong Q. Nguyen and Oded Regev. *Learning a Parallelepiped: Cryptanalysis of GGH and NTRU Signatures*, volume 4004 of *Lecture Notes in Computer Science*, pages 271–288. Springer Berlin / Heidelberg, 2006.

[PS10] Clément Pernet and William Stein. Fast computation of hermite normal forms of random integer matrices. *Journal of Number Theory*, 130(7):1675–1683, July 2010.

[PSW08] Thomas Plantard, Willy Susilo, and Khin Than Win. *A Digital Signature Scheme Based on CVP_∞*, volume 4939/2008 of *Lecture Notes in Computer Science*, pages 288–307. Springer Berlin / Heidelberg, 2008.

[Ram95] J. Ramanujam. Beyond unimodular transformations. *Journal of Supercomputing*, 9:365–389, 1995.

[RPS11] Michael Rose, Thomas Plantard, and Willy Susilo. Improving BDD cryptosystems in general lattices. In *Information Security Practice and Experience, 7th International Conference, ISPEC 2011*, volume 6672 of *Lecture Notes in Computer Science*, pages 152–167. Springer Berlin Heidelberg, 2011.

[S+97] Martin Schönert et al. *GAP - Groups, Algorithm, Programming – version 3 release 4 patchlevel 4*. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1997. 1571 pages, available from `http://www.gap-system.org/`.

[S+07] William A. Stein et al. *Sage Mathematics Software (Version 2.7)*. The Sage Development Team, 2007. available from `http://www.sagemath.org/`.

[Sho09] Victor Shoup. NTL: A library for doing Number Theory, August 2009. http://www.shoup.net/ntl/.

[Ste] Allan Steel. Hermite normal form timings page. http://magma.maths.usyd.edu.au/users/allan/mat/hermite.html.

[Sto96] Arne Storjohann. Computing hermite and smith normal forms of triangular integer matrices. *Linear Algebra Applications*, 282:25–45, 1996.

[SV10] N. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Nguyen and David Pointcheval, editors, *Public Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer Berlin / Heidelberg, 2010.

[Tit30] E. C. Titchmarsh. A divisor problem. *Rend. Circ. Math. Palermo*, 54:414–429, 1930.

[vzGG99] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[Wei] Eric W. Weisstein. Legendre's conjecture. From Mathworld – A Wolfram Web Resource http://mathworld.wolfram.com/LegendresConjecture.html.

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 887–898, New York, NY, USA, 2012. ACM.