Factorization Properties of Lattices Over the Integers*

Azaria Paz and Mody Lempel

Department of Computer Science
Technion—Israel Institute of Technology
32000 Haifa, Israel

Submitted by Raphael Loewy

ABSTRACT

Let \( A \) be a nonsingular \( n \times n \) matrix over the integers. \( L = L(A) \) denotes the lattice whose elements are combinations with integer coefficients of the rows of \( A \). \( L \) is cyclic if it can be defined in the modular form \( L = \{ x = (x_i) : \Sigma a_i x_i \equiv 0 \pmod{d} \} \) where the \( a_i \)'s and \( d \) are integers and \( 0 \leq a_i < d \). Let \( L, L_1, L_2(B) \) be lattices over the integers. \( L = L_1 L_2 \) is a factorization of \( L \) if every element of \( L \) is a combination of the rows of \( B \) such that the vector of combination coefficients is in \( L_1 \), and \( B \) is a nonsingular \( n \times n \) matrix. The following results are proved: Every lattice can be expressed as a product of cyclic factors in polynomial time; every cyclic lattice can be factored into "simple" (term explained in the text) factors in polynomial time; every simple lattice can be factored into "prime" factors in polynomial time if a prime factorization of the determinant of its basis is given. In addition we provide polynomial algorithms for the following problems: transform a cyclic lattice given by a basis into a modular form and vice versa; find a basis of a finite modular lattice, given in modular form.

1. INTRODUCTION AND MOTIVATION

A lattice in \( \mathbb{R}^n \), the Euclidean \( n \)-space over the integers, can be defined as a set of \( n \)-dimensional vectors with integer coordinates which is closed under addition and subtraction.

*Most of the results shown in this paper are based on part of the M.Sc. thesis of the second author, done under the supervision of the first author and submitted to the Senate of the Technion.
Such lattices have a rich structure and have many applications. They have been studied in connection with the general subject called "geometry of numbers" [5, 7]. In the past 10 years or so, some algorithmic aspects of lattices have been investigated in connection with various important problems, e.g. factoring polynomials with rational coefficients [9], integer programming [10], finding the distance of codes [4, 11, 12], etc.

A basis for a lattice is defined to be a set of linearly independent elements of the lattice such that every element of the lattice is a linear combination, with integer coefficients, of those elements. Usually a lattice is defined by a matrix with integer entries, whose rows form its basis. Reduced bases have been defined in several ways (e.g., a base is reduced if the product of the lengths of its vectors is minimized), and base reduction algorithms have been introduced [9]. Another important algorithmic problem considered in the literature is the problem of finding the shortest vector of a lattice [6, 8].

A lattice in \( n \)-space is of full rank if the linear space generated by its elements is of dimension \( n \).

The following definitions will be used in the sequel.

**Definition 1.** A lattice of full rank is called *cyclic* if it can be represented in the form

\[
L(a_1, \ldots, a_n, d) = \left\{ x = (x_i) : \sum_{i=0}^{n} a_i x_i \equiv 0 \pmod{d}; \quad \gcd(a_1, \ldots, a_n, d) = 1; \ 0 < a_i < d, a_i, d, x_i \in \mathbb{Z}, 1 < i \leq n \right\}.
\]

A representation as above for \( L \) will be called a *cyclic representation*. Cyclic lattices and their representation have been investigated by Paz and Schnorr [13]. Their cyclic representation reduces the number of parameters needed for representing \( L \) from \( n^2 \) parameters (required for specifying a basis) to \( n + 1 \) parameters, namely \( a_1, \ldots, a_n, d \).

**Definition 2.** A *finite* lattice \( L_d \) is called *modular* if it can be defined as below:

\[
L_d = \left\{ x = (x_i) : \sum_{i=1}^{n} a_i x_i \equiv 0 \pmod{d}, \quad \gcd(a_1, \ldots, a_n, d) = 1; \ 0 \leq a_i, x_i < d, a_i, d, x_i \in \mathbb{Z}, 1 \leq i \leq n \right\}.
\]
A representation as above for \( L_d \) will be called a \textit{modular representation}.

Notice that the finiteness of a modular lattice is reflected in the condition that \( 0 \leq x_i < d \), for all \( x = (x_1, \ldots, x_n) \in L_d \).

For a given modular lattice \( L_d \), a \textit{modular basis} for \( L_d \) is a set of elements in \( L_d \) such that every element in \( L_d \) is a modular combination of its elements with modulus \( d \) and the coefficients of combinations \( c_i \) satisfy \( 0 \leq c_i < d \).

**Definition 3.** Let \( L, L_1, L_2 \) be lattices (over the integers). \( L_1L_2 \) is a factorization of \( L \) (notation: \( L = L_1L_2 \)) if and only if

\[
L = \{ w = w_1A : w_1 \in L_1; \text{ the rows of } A \text{ are a basis for } L_2 \}.
\]

Three polynomial-time factorization algorithms are given in this paper. The first algorithm factors a lattice, given by a basis \( B \), into cyclic factors. The factor lattices produced by the algorithm are provided explicitly in both forms, by a basis and by a vector \( a = (a_i) \) of coefficients defining the corresponding modular equation \( \sum a_ix_i \equiv 0 \pmod{d} \).

The second algorithm factors a cyclic lattice, given in modular form, into simple and cyclic lattices. A simple lattice is a lattice which can be defined by a basis matrix which is equal to the unit matrix except for one column, whose diagonal element is equal to a divisor \( d_1 \) of \( d \) (the modulus of the modular equation defining the lattice at input) and whose off-diagonal elements are nonnegative and less than \( d_1 \). The factor lattices produced by this algorithm are also provided in both forms, by a basis matrix and by a vector of coefficients.

Both algorithms can be applied in sequence, resulting in the factorization of a given lattice into cyclic and simple lattices. If \( B \) is a basis of the given lattice and \( B_i \) are bases for its factors, then the determinant of \( B \), in absolute value, is equal to the product of the determinants of the \( B_i \)'s in absolute value—as should be expected. On the other hand, the algorithms do not require, or provide, a factorization of \( |B| \) into prime factors.

The first algorithm can be used in order to get a modular representation for a cyclic lattice when it is given by a basis. The second algorithm provides a basis for a cyclic lattice when given in a modular form. The two algorithms can be used therefore in order to transform one form of representing lattices into the other (by a basis or by a modular equation).

Assume that \( B \) is a simple basis for a lattice \( L \), and assume that a factorization of \( |B| = d \) into prime factors is given. A third factorization algorithm is shown in Section 8, factoring \( L \) into simple factor lattices such that if \( B_i \) is a basis for \( L_i \) then \( |B_i| \) is prime.
This algorithm requires the factorization of $d$ into prime factors. If such a factorization is provided for $d = |B|$ and $B$ is a basis for a general lattice $L$, then, by applying all three algorithms in sequence, we can factor $L$ into a sequence of factor lattices $L_i$ such that each $L_i$ is cyclic and simple and the determinant of its basis is equal to a prime factor of $d$.

In the last section of the paper we show how to use the second algorithm in order to find a modular basis for any given modular lattice, except for two degenerate cases [$a = (a_i)$ has a single nonzero element, or it has exactly two such elements and one is the modular negative of the other].

The above algorithms provide a factorization into simple factors of any given lattice of full rank. It is hoped that such a factorization will provide some new techniques for dealing with the problems mentioned at the beginning of this introduction (basis reduction, finding short vectors, etc.).

2. PRELIMINARIES

Lattices considered in this paper are sets of vectors with integer coordinates in $n$-dimensional Euclidean space, closed under addition and subtraction. A lattice is of full rank if the linear space generated by its elements is of dimension $n$. A basis for such a lattice is a set of $n$ vectors which belong to the lattice and such that every vector in the lattice is a combination, with integer coefficients, of its elements. If a lattice $L$ of full rank is given by a basis, then the vectors forming the basis will be given as rows of an $n \times n$ nonsingular matrix $B$ and the lattice will be denoted by $L(B)$.

Let $L_1$ and $L_2$ be lattices. $L_1$ refines $L_2$ if $L_2 \subseteq L_1$.

Let $L_1$ and $L_2$ be lattices of full rank, and let $B_1$ and $B_2$ be corresponding bases. $L_1$ is a right factor of $L_2$ if $CB_1 = B_2$ for some matrix $C$ with integer entries.

Trivially $L_1$ is a right factor of $L_2$ if and only if $L_1$ refines $L_2$.

A matrix will be called unimodular if it has integer entries and its determinant is equal to $\pm 1$. Trivially, the inverse of a unimodular matrix is unimodular.

A lattice may have many bases, but, as is easy to prove and well known, $B_1$ and $B_2$ are bases for the same lattice iff there exists a unimodular matrix $U$ such that $B_1 = U B_2$. It follows that the determinants of all bases of a given lattice are equal to one another. Thus, the determinant of a basis of a lattice $L$ is an invariant of $L$.

In the sequel we shall consider lattices of full rank unless otherwise specified.
3. A FACTORIZATION ALGORITHM

Let \( L \) be a lattice given by a basis \( B \). Our first goal is to provide an algorithm for factoring \( L \) into a sequence of cyclic lattices \( L_1, \ldots, L_k \). For each \( i, 1 \leq i \leq k \), the algorithm will provide a sequence of integers \( a_{i1}, \ldots, a_{in}, d_i \) and a basis \( B_i \) such that \( L_i = L(a_{i1}, \ldots, a_{in}, d_i) = L(B_i) \) and such that \( |d| = |d_1 \cdots d_k| \), where \( d \) is the determinant of \( B = B_k B_{k-1} \cdots B_1 \).

Notice that if \( |B| = \pm 1 \) and \( B \) has integer entries, then \( B^{-1} \) is also a matrix with integer entries, implying that \( B^{-1}B = I \) is a basis for the same lattice. Such a lattice must therefore coincide with the lattice of all vectors with integer coordinates. This lattice will be called the natural lattice.

The natural lattice will be considered as a cyclic lattice, by definition, since it can be defined in the form

\[
Z^n = \left\{ (x_1, \ldots, x_n) : \sum x_i \equiv 0 \pmod{1} \right\}
\]

Notice that the coefficients of \( x_i \) in the above summation are all equal to 1 and do not satisfy the requirement that \( a_i < d_i \) (since \( a_i = 1 = d_i \)).

**Algorithm CF (Cyclic factorization).**

1. Given \( B \), find \( d := |B| \), set \( d := |d| \), \( U := B \).
2. If \( d = 1 \) return \( (a_1, \ldots, a_n) = (1, \ldots, 1), d := 1 \), \( B := I \) \( \{L(B) = L(I) = L(1, \ldots, 1); L(B) \text{ is cyclic by definition}\} \) else continue.
3. Compute the matrix \( dU^{-1} \); then reduce its entries to nonnegative integers modulo \( d \). Denote the resulting matrix by \( W \).
4. Let \( w^T = (w_1, \ldots, w_n)^T \) be any nonzero column of \( W \). (Such a column exists; see Lemma 1.) Compute \( g = \gcd(w_1, \ldots, w_n, d) \); set \( i := 1 \).
5. While \( g > 1 \) do
   
   \[ \begin{array}{c}
   \text{begin} \\
   5.1 \text{ Reset} \ (w'_1, \ldots, w'_n, d') := (1/g)(w_1, \ldots, w_2, d).
   \\
   5.2 \text{ Find a unimodular matrix} \ R' \text{ such that}
   \\
   \begin{bmatrix}
   w'_1 \\
   \vdots \\
   w'_n \\
   d'
   \end{bmatrix} = \begin{bmatrix}
   0 \\
   \vdots \\
   0 \\
   1
   \end{bmatrix}
   \end{array} \]
   
   (Such a matrix can always be found; see appendix.)
5.3 Denote by \( R \) the matrix derived from \( R' \) by removing its last row and its last column.
5.4 Output \( (a_{i1}, \ldots, a_{in}, d_i) := (w'_1, \ldots, w'_n, d'), B_i = R; \{L_i = L_i(B_i) = L_i(a_{i1}, \ldots, a_{in}, d_i)\} \).
5.5 Find a matrix $V$ with integer entries such that $VR = U$.

5.6 Reset $W := (1/d')RW \mod g$; $U := V := UR^{-1}$; $d := g$; $i := i + 1$.

5.7 Let $w^T = (w_1 \cdots w_n)^T$ be any nonzero column of $W$. Compute $g := \gcd(w_1, \ldots, w_n, d)$

End (while);

6. Output $(a_{i1}, \ldots, a_{in}, d_i) := (w_1, \ldots, w_n, d); B_i = U; \{L_i = L_i(w_1, \ldots, w_n, d) = L_i(B_i)\}.

End of algorithm.

4. PROOF OF CORRECTNESS AND COMPLEXITY

**Lemma 1.** The matrix $W$ as defined in step 3 contains at least one nonzero column, and all its entries are integers.

**Proof.** By Cramer's rule every entry $U^{-1}$ has the form $z/d$. Given that the absolute value of the determinant of $U$ is $|d| > 1$, the absolute value of the determinant of $U^{-1}$ is less than 1, implying that at least one entry in $U^{-1}$ is not an integer. Assume this entry to be $a/b$. In $W$ this entry changes into $d(a/b) \mod |d|$. If in $W$ this corresponding entry is equal to zero, then $d(a/b) = kd$ for some nonzero integer $k$. But this implies that $a/b$ is an integer, contrary to our assumption.\[\square\]

Lemma 1 justifies step 4 of the algorithm.

**Lemma 2.** The lattice $L(w_1, \ldots, w_n, d)$ refines the lattice $L(B)$, where $B$ and $w_1, \ldots, w_n$ are as defined in steps 1 to 4 of the algorithm.

**Proof.** Assume that the $j$th column of $W$ is a nonzero column, and let $y^T$ be the $j$th column of $dU^{-1}$. Then $Uy^T = de^T_j$, where $e^T_j$ is the vector whose $j$th entry is equal to one and all other entries equal to zero. Let $y = (y_1 \cdots y_n)$, and set $w_i = y_i \mod d$. Then $w^T = (w_1 \cdots w_n)^T$ is the $j$th column of $W$, and we have that $Uw^T = dk^T$, where $k = (k_i)$ is a vector of integers. As $U = B = [b_{ij}]$ the above equality implies that $\sum_j b_{ij}w_j = k_i d$, $1 \leq i \leq n$, so that the elements of the basis of $L(B)$ represented by the rows of $B$, and any combination with integer coefficients of those elements are in $L(w_1, \ldots, w_n, d)$.\[\square\]

**Lemma 3.** Let $R$ be the matrix defined in step 5.3 of the algorithm. Then

$L(R) = L(w'_1, \ldots, w'_n, d')$. 

FACTORIZATION OF INTEGER LATTICES 253

Proof. Let \((b'_{i,1}, \ldots, b'_{i,n+1})\) be the \(i\)th row, \(1 \leq i \leq n\), of \(R'\). Then, by the definition of \(R'\), we have that \(\sum_{j=1}^{n} b'_{ij} w_j + b'_{in} d = 0\), implying that \((b_{i1}, \ldots, b_{in})\), the \(i\)th row of \(R\), is in \(L(w'_1, \ldots, w'_n, d')\).

To prove the other direction notice first that \(R'\) is invertible over the integers by its definition, and that the last column of \((R')^{-1}\) must be equal to \((w'_1 \cdots w'_n, d')^T\), again by definition (see step 5.2). Let \((c_1 \cdots c_n) = c\) be any vector in \(L(w'_1, \ldots, w'_n, d')\). Then \(c\) satisfies \(\sum_{j=1}^{n} c_j w' + kd' = 0\) for some integer \(k\). Therefore \((c_1 \cdots c_n k)(R')^{-1} = (m_1 \cdots m_n 0)\), where \(k\) and the \(m_i\)'s are integers, since the entries of \((R')^{-1}\) are integers and its last column is \((w'_1 \cdots w'_n d')\).

It follows that \((c_1 \cdots c_n k) = (m_1 \cdots m_n 0)R'\), and the vector \(c\) is shown to be a combination, with integer coefficients, of the rows of \(R\) (the coefficient of the last row of \(R'\) in the combination is zero, and the last column of \(R'\) is deleted). ■

REMARK. Finding a matrix \(R'\) as required in step 5.2 can be done via any extended Euclidean algorithm of Blankenship type (see appendix), given that \(\gcd(w'_1, \ldots, w'_n, d') = 1\), as is the case after step 5.1.

**Lemma 4.** Let \(R\) be the matrix defined in step 5.3 of the algorithm. Then \(|R| = d' = d/g\).

Proof. By definition

\[
R' = \begin{bmatrix}
R & a^T \\
b & c
\end{bmatrix},
\]

where \(a^T\) and \(b\) are vectors of integers and \(c\) is an integer. Consider the equation \(R'x = (0 \cdots 0 1)^T\). By Cramer's rule we have that

\[
x_{n+1} = \frac{1}{|R'|} \begin{bmatrix} 0 \\ R \\ 0 \\ b \end{bmatrix} = |R|,
\]

since \(|R'| = 1\).

By definition (step 5.2)

\[
R' \begin{bmatrix} w'_1 \\ \vdots \\ w'_n \\ d' \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix},
\]

so that \((x_1 \cdots x_{n+1})^T = (w'_1 \cdots w'_n d')\) satisfies the equation \(R'x = (0 \cdots 0 1)^T\).
It follows that \( x_{n+1} \) must be equal to \( d' \), and it must be equal to \(|R|\) (by Cramer’s rule), so that \(|R| = d'\).

**Lemma 5.** Let \( V \) be the matrix defined in step 5.5 of the algorithm. Such a matrix can be found, and it has the properties that \( gV^{-1} \mod g = (1/d')RW \mod g \), where \( W \) is defined in step 3 and \( R \) is defined in step 5.3, and it has integer entries.

**Proof.** \( U = B \) (step 1) and \( VR = U \). From \(|R| = d'\), \(|U| = |B| = d\) we get that \(|V| = d/d' = g\). Trivially \( L(w_1, \ldots, w_n, d) = L(w'_1, \ldots, w'_n, d') \), by the definitions and by the fact that \((w'_1, \ldots, w'_n, d') = (1/g)(w_1, \ldots, w_n, d)\), where \( g = \gcd(w_1, \ldots, w_n, d) \). From Lemmas 2 and 3 we get therefore that \( L(R) = L(w'_1, \ldots, w'_n, d') \) refines \( L(U) = L(B) \). It follows that \( R \) is a right factor of \( U \), so that a matrix \( V \) with integer entries such that \( VR = U \) can be found.

Now, since \( RU^{-1} = V^{-1} \) it follows that \((g/d)RDU^{-1} = gV^{-1} \). Therefore \( gV^{-1} \mod g = (g/d)RDU^{-1} \mod g = (1/d')RW + gRE \mod g = (1/d')RWg \mod g \), where \( E \) is a matrix with integer entries (by the definition of \( W \)), and \( d = d'g \).

**Remark.** Since \( gV^{-1} \) is a matrix with integer entries, it follows that \((1/d')RW \) has the same property and, by Lemma 1, \((1/d')RW \mod g \) must have a nonzero column. The correctness of the algorithm follows from the above lemmas. At iteration \( i \) of the while loop the lattices \( L(U) \) is factored into a cyclic right factor \( L(R) = L(a_{i1}, \ldots, a_{in}, d_i) \) and a left factor \( L(V) \), which is the \( L(U) \) of the next iteration, and \(|V| \) divides \(|U|\).

After at most \( \log_2 |B| \) iterations the algorithm halts with a cyclic leftmost factor (step 6). All through the computation the algorithm produces matrices and vectors with integer entries.

**Lemma 6.** The number of arithmetical operations involved in the CF algorithm is \( O((n^3 + \log d) \log d) \), and the magnitude of the intermediate results is \( O(\max(n^2M^{2n}, nd^n)) \), where \( M \) is the maximal entry, in absolute value, of the matrix \( B \) at input.

**Proof.** Finding the determinant of \( B \) (step 1) and computing \( dU^{-1} \) (step 3) can be done in \( O(n^3) \) operations with intermediate results bounded by \( n^2M^{2n} \); see the algorithm of Edmonds [3].

The entries of the vector whose gcd is to be found at step 4 are bounded by \( d \). The number of operations involved in the finding of the gcd is therefore \( O(\log d + n) \) with the intermediate results bounded by \( d \); see Bradley [2].

The number of iterations of the while loop 5 is bounded by \( \log d \).
Step 5.1 is $O(n)$.
Step 5.2 is $O(n^2 + \log d)$ with intermediate results bounded by $nd^n$; see appendix.
Step 5.5, 5.6, and 5.7 are $O(n^3)$ with intermediate results bounded by $nd^3$.

**Remark.** The vectors $(a_{i1}, \ldots, a_{in}, d_i)$, output at step 6, have nonnegative entries and $a_{ij} < d_i$ for all $i$ and $j$.

5. A SECONDARY FACTORIZATION ALGORITHM

A matrix over the integers whose determinant is equal to $d$ will be called $d$-simple if it is a unit matrix except for one column whose diagonal element equals $d$ and whose off-diagonal elements are all nonnegative and less than $d$.

We provide now an algorithm for factoring any given cyclic lattice $L(a_1, \ldots, a_n, d)$ into a sequence of cyclic lattices $L_1, \ldots, L_k$ such that the lattice $L_i$ in this sequence is represented by a $d$-simple matrix whose rows are its basis, and $d = d_kd_{k-1}\cdots d_1$.

We precede the algorithm with a simple procedure.

**Definition.** Given two integers $a, b > 0$, the greatest uncommon divisor of $a$ with regard to $b$ [notation $\gcd(a, b)$] is the greatest integer $a_1$ such that $a_1$ divides $a$ and $\gcd(a_1, b) = 1$.

Notice that this definition is not symmetric with regard to $a$ and $b$.

To find the gcd of $a$ with regard to $b$ we can use the following procedure.

**Procedure** $\gcd(a, b)$.

1. Compute $g := \gcd(a, b)$;
2. While $g > 1$ do begin
   a := $a/g$;
   $g := \gcd(a, b)$
   end;
3. Output $a$.

**Remark.** The complexity of finding the gcd is logarithmic in the magnitude of the numbers involved, and the number of iterations of the while loop is logarithmic in $a$. The complexity of this algorithm is therefore $O(\log^2 a)$. 
Notice that if \( \text{gcd}(a, b) = u \) then \( \text{gcd}(u, b) = 1 \).

The main algorithm is now described.

**ALGORITHM SF** (Simple factorization). Input \((a_1^{(0)}, \ldots, a_n^{(0)}, d^{(0)}) := (a_1, \ldots, a_n, d)\) such that \(0 \leq a_l < d\) for \(1 \leq l \leq n\), some \(a_l\) are positive for \(1 \leq l \leq n\), and \(\text{gcd}(a_1, \ldots, a_n, d) = 1\). (These properties hold true for the vectors output by the CF algorithm at step 6.)

1. Set \(i := 1, j := 1\);
2. While \(a_i^{(j-1)} = 0\) or \(\text{gcd}(d^{(j-1)}, a_i^{(j-1)}) = 1\) set \(i := i + 1\);
3. Set \(d_j := \text{gcd}(d^{(j-1)}, a_i^{(j-1)})\), output \(d_j\) (now \(d_j > 1\) but \(\text{gcd}(d_j, a_i^{(j-1)}) = 1\), by definition);
4. Define the \(d_j\)-simple matrix \(A(j) = [s_{ij}^{(j)}]\) as the unit matrix except for its \(i\)th column, whose diagonal element is equal to \(d_j\) and whose off-diagonal elements are \(s_{ii}^{(j)} = -(a_l^{(j-1)}/a_i^{(j-1)}) \mod d_j, 1 \leq l \leq n, l \neq i\); output \(A(j)\);
5. Reset
   \[
   a_i^{(j)} := \frac{1}{d_j} \left( s_{ii}^{(j)} a_i^{(j-1)} + a_i^{(j-1)} \right), \quad 1 \leq l \leq n, \quad l \neq i
   
   a_i^{(j)} := a_i^{(j-1)} \quad \left\{ = \frac{1}{d_j} s_{ii}^{(j-1)}, \text{ since } s_{ii} = d_j \right\}
   
   d^{(j)} := d^{(j-1)}/d_j;
   
   Output \(a^{(j)} = (a_1^{(j)}, \ldots, a_n^{(j)})\) \(\{a^{(j)} = (1/d_j)A(j)a^{(j-1)}\};
6. If \(d^{(j)} > 1\) then set \(j := j + 1, i := i + 1\) and go to 2;
7. Halt.

6. **PROOF OF CORRECTNESS**

The correctness of the algorithm follows from the lemmas proven below.

The number of iterations of the algorithm is finite, as will be shown in the sequel.

**LEMMA 7.** For \(0 \leq j \leq k\), where \(k\) is the number of iterations of the algorithm:

(a) \(\text{gcd}(a_1^{(j)}, \ldots, a_n^{(j)}, d^{(j)}) = 1\).

(b) For \(j \geq 1, 1 \leq l \leq n\), if \(\text{gcd}(d^{(j-1)}, a_l^{(j-1)}) = 1\) then \(\text{gcd}(d^{(j)}, a_l^{(j)}) = 1\), and

(c) if \(a_l^{(j-1)} = 0\) then \(a_l^{(j)} = 0\).
Proof. Part (a) of the lemma is proved by induction. It is true for \( j = 0 \) by assumption. Assume by induction that \( \gcd(a^{(j-1)}, d^{(j-1)}) = 1 \). If \( \gcd(a^{(j)}, d^{(j)}) = g > 1 \), then, by definition (see step 4) \( d_j a_i^{(j)} = s_{li}^{(j)} a_i^{(j-1)} + a_i^{(j-1)} \) and \( a_i^{(j)} = a_i^{(j-1)} \) for some \( i \). Thus \( g \) divides \( a_i^{(j)} \) and \( a_i^{(j-1)} = a_i^{(j)} \), implying that \( g \) divides \( a_i^{(j-1)} \), \( 1 \leq l \leq n \). Also \( d^{(j-1)} = d^{(j)} d_j \), so that if \( g \) divides \( d^{(j)} \) then it divides \( d^{(j-1)} \). Thus \( g \) divides all the entries of \( a^{(j-1)} \), and \( g \) divides \( d^{(j-1)} \), contradicting the assumption that \( \gcd(a^{(j-1)}, d^{(j-1)}) = 1 \).

Part (b) of the lemma follows from the following considerations:

From the definitions of \( d_j \) and \( d^{(j)} \) (steps 3 and 5) it is clear that \( d_j \) and \( d^{(j)} \) are relatively prime, and it is easy to see that if \( \gcd(d^{(j-1)}, a_i^{(j-1)}) = 1 \) then all the prime factors of \( d^{(j-1)} \) are factors of \( a_i^{(j-1)} \). For the given iteration \( j - 1 \) let \( i \) be the index for which the equalities below hold:

\[
d_j a_i^{(j)} = s_{li}^{(j)} a_i^{(j-1)} + a_i^{(j-1)}, \quad a_i^{(j)} = a_i^{(j-1)}, \quad l \neq i.
\]

By the definition and by our assumption, \( d_j \) contains all the prime factors of \( d^{(j-1)} \) which are not factors of \( a_i^{(j-1)} \), so that all the prime factors of \( d^{(j)} = d^{(j-1)} d_j \) are factors of \( a_i^{(j-1)} \), and are not factors of \( d_j \). Also all the prime factors of \( d^{(j)} \) are factors of \( a_i^{(j-1)} \), \( l \neq i \) [since \( d^{(j)} \) divides \( d^{(j-1)} \) and we assumed that \( \gcd(d^{(j-1)}, a_i^{(j-1)}) = 1 \)].

So all the prime factors of \( d^{(j)} \) divide the right-hand side of the equation defining \( d_j a_i^{(j)} \). Since the prime factors considered do not divide \( d_j \), they must divide \( a_i^{(j)} \) in the left-hand side of the equation. It follows that \( \gcd(d^{(j)}, a_i^{(j)}) = 1 \) for \( l \neq i \). If \( l = i \) then \( \gcd(d^{(j-1)}, a_i^{(j-1)}) = d_j > 1 \), by our assumption, and therefore the antecedent of property (b) does not hold for \( l = i \). This completes the proof of property (b).

The prove part (c) of the lemma notice that if \( a_i^{(j-1)} = 0 \) then \( s_{li}^{(j)} = 0 \) (step 4), which implies that \( a_i^{(j)} = 0 \) (step 5) for \( l \neq i \), where \( i \) is the index satisfying \( a_i^{(j)} = a_i^{(j-1)} \) at the \( j \)th iteration. It follows from step 2 that \( a_i^{(j-1)} \neq 0 \) for \( l = i \), so that the premise of property (c) does not hold for \( l = i \). The proof of lemma 7 is now complete.

Consider now the \( j \)th iteration of the algorithm. Since \( \gcd(a^{(j-1)}, d^{(j-1)}) = 1 \) [by property (a) of Lemma 7], it follows that every prime factor \( p \) of \( d^{(j-1)} \) fails to divide some entry \( a_i^{(j-1)} \neq 0 \) of \( a^{(j-1)} \), implying that \( \gcd(d^{(j-1)}, a_i^{(j-1)}) \geq 1 \) for that particular \( l \).

Moreover, the smallest index \( l \) satisfying \( \gcd(d^{(j-1)}, a_i^{(j-1)}) > 1 \) is bigger than the smallest index \( l' \) satisfying \( \gcd(d^{(j-2)}, a_i^{(j-2)}) > 1 \), as follows from properties (b) and (c) of Lemma 7 and from the fact that \( \gcd(d^{(j-1)}, a_i^{(j-1)}) = 1 \).
It follows that step 3 of the algorithm provides a divisor $d_j > 1$ of $d^{(j-1)}$, provided that $d^{(j-1)} > 1$, so that in the end $d = d^{(0)} = d_1 d_2 \cdots d_k$, where $k$ is the number of iterations.

It is also clear from the above considerations that the number of iterations is bounded by $n$ (the running index $n$ in step 2 grows at every iteration and $i \leq n$) and is also bounded by $\log_2 d^{(0)}$ ($d(j)$ decreases by a factor of at least 2 at every iteration). To complete the correctness proof we must show that the lattices $L(A^{(j)})$, where $A^{(j)}$ are the $d_j$-simple matrices output at step 4, provide a decomposition of $L(a^{(0)}, d^{(0)})$. This is done in the next lemmas.

Define the lattices $L_j$ and $C_j$ as $L_j = L(a^{(j-1)}, d_j)$, $j \geq 0$, $C_j = L(a^{(j)}, d^{(j)})$, $j \geq 0$, where the $d_j$'s are as defined in step 3 of the algorithm.

**Lemma 8.** $L_j = L(A^{(j)})$, where $A^{(j)}$ is the matrix defined at step 4 of the algorithm.

**Proof.** Extend the matrix $A^{(j)}$ into an $(n + 1) \times (n + 1)$ matrix $B^{(j)}$ which satisfies the equation below:

$$
\begin{align*}
B^{(j)} \begin{bmatrix}
1 & s_{1,1} & \cdots & s_{1,i} & k_1 \\
& \ddots & \ddots & \ddots & \ddots \\
& & 1 & s_{i-1,i} & K_{i-1} \\
& & & d_j & -a_i^{(j-1)} \\
& & & s_{i+1,i} & k_{i+1} \\
& & & \ddots & \ddots \\
& & & & s_{n,i} \\
x & & & & y
\end{bmatrix}
&= \begin{bmatrix}
\begin{bmatrix} a_1^{(j-1)} \\
\vdots \\
a_n^{(j-1)} \end{bmatrix} \\
\end{bmatrix} = \begin{bmatrix}
0 \\
\vdots \\
1
\end{bmatrix} \quad (\ast)
\end{align*}
$$

The matrix $B^{(j)}$ is constructed as follows: The upper $n \times n$ main diagonal block of $B^{(j)}$ is the matrix $A^{(j)}$; $x$ and $y$ are two integers such that

$$
x a_i^{(j-1)} + y d_j = 1, \quad b_{n+1,i}^{(j)} = x, \quad b_{n+1,n+1}^{(j)} = y.
$$

Such integers exist because, by definition,

$$
gcd \left( a_i^{(j-1)}, d_j \right) = 1,
$$

$$
b_{n+1,i}^{(j)} = 0 \quad \text{for} \quad l \neq i \text{ and } l \neq n + 1,
$$

$$
b_{n+1,n+1}^{(j)} = y.
$$
$k_l = -(1/d_l)(a_l^{(j-1)} + s_l^{(j)}a_i^{(j-1)})$, $l \neq i$. By the construction of $A^{(j)}$ we have that $a_l^{(j-1)} + s_l^{(j)}a_i^{(j-1)} \equiv 0 \pmod{d_l}$ and therefore $k_l$ is an integer.

It is easy to verify now that the matrix $B^{(j)}$ constructed as above satisfies the equation $(*).$

It is also easy to verify that the determinant of $B^{(j)}$ is equal to $yd_j + xa_l^{(j-1)} = 1$ (develop the determinant by its last row) and $B^{(j)}$ is therefore a unimodular matrix, implying that $(B^{(j)})^{-1}$ is a matrix with integer entries. Moreover, since $B^{(j)}B^{(j)} = I$ and $B^{(j)}$ satisfies the equation $(*),$ the last column of $(B^{(j)})^{-1}$ must equal $(a_1^{(j-1)} \ldots a_n^{(j-1)}d_j)^T.$

Let $w = (w_1 \ldots w_n)$ be a vector in $L_j.$ Then $(w, a^{(j-1)}) \equiv 0 \pmod{d_j}$ where $(,) \text{ denotes scalar multiplication of vectors. There is therefore an integer } k \text{ such that } ((w_1 \ldots w_nk), (a_1^{(j-1)} \ldots a_n^{(j-1)}d_j)) = 0, \text{ and since } (a_1^{(j-1)} \ldots a_n^{(j-1)}d_j)^T \text{ is equal to the last column of } (B^{(j)})^{-1}, \text{ we have that}$

$$(w_1 \ldots w_nk)(B^{(j)})^{-1} = (m_1 \ldots m_n0),$$

where the $m_i$'s are integers. Therefore $(w_1 \ldots w_nk) = (m_1 \ldots m_n0)B^{(j)},$ and if the last coordinate is ignored, this reduces to $(w_1 \ldots w_n) = (m_1 \ldots m_n)A^{(j)},$ since only the $n \times n$ main diagonal block of $B^{(j)},$ which is equal to $A^{(j)},$ contributes to the above equation.

We have thus shown that if $w$ is in $L_j$ then $w$ is in $L(A^{(j)}).$ The other direction is trivial, since all the rows of $A^{(j)}$ are in $L_j$ by construction (step 4 in the algorithm).

Let $L, L_1, L_2$ be lattices over the integers, and let $L_1L_2$ be a factorization of $L$ (see Definition 3 in Section 1).

**Lemma 9.** For $0 \leq j \leq k - 1,$ where $k$ is the number of iterations of the SF algorithm,

$$C_j = C_{j+1}L_{j+1}.$$

*Proof.* By Lemma 8, $L_{j+1} = L(A^{(j+1)}).$

Assume first that $w \in C_{j+1}.$ We show that $wA^{(j+1)} \in C_j$: we have

$$\left\langle wA^{(j+1)}, a^{(j)} \right\rangle = \left\langle w, A^{(j+1)}a^{(j)} \right\rangle = \left\langle w, d_{j+1}a^{(j+1)} \right\rangle$$

(by step 5 in the SF algorithm). The above scalar product is equal to $d_{j+1}(w, a^{(j+1)}),$ which by our assumption is equal to $d_{j+1}kd^{(j+1)} = kd^{(j)}$ for some integer $k.$ Thus $(wA^{(j+1)}, a^{(j)}) \equiv 0 \pmod{d^{(j)}},$ as required.

Assume now that $wA^{(j+1)} \in C_j,$ i.e., $(wA^{(j+1)}, a^{(j)}) = kd^{(j)}$ for some integer
k. We show that \( w \in C_{j+1} \): We have

\[
\left\langle w, a^{(j-1)} \right\rangle = \frac{1}{d_{j+1}} \left\langle w, d_{j+1} a^{(j+1)} \right\rangle = \frac{1}{d_{j+1}} \left\langle w, A^{(j+1)} a^{(j)} \right\rangle
\]

(by step 5 in the SF algorithm). The above scalar product is equal to \((1/d_{j+1}) \langle w A^{(j+1)}, a^{(j)} \rangle\), which by our assumption is equal to \((1/d_{j+1}) \ k d^{(j)} = k a^{(j+1)}\). This implies that \( w \in C_{j+1} \), as required.

The values \( a_i \) computed in step 5 of the algorithm are integers. This follows from the definitions:

\[
c_{sf,i}^{(j-1)} = a_i^{(j-1)} + (-a_i^{(j)} \mod d_j)
\]

for some integer \( k_l \). As \( \gcd(a_i^{(j-1)}, d_j) = 1 \), \( a_i^{(j)} \) is uniquely defined: \( a_i^{(j)} = k_l \).

The correctness of the algorithm is now implied by the above two Lemmas 8 and 9. After at most \( k < \log_2 d \) iterations the algorithm halts. At the \( j \)th iteration the algorithm outputs \( d_j, a^{(j)}, A^{(j)} \). We have shown that \( d = d_1 \cdots d_k \); that \( L = L_k L_{k-1} \cdots L_1 \) (where \( L \) is the lattice at input), as follows from Lemma 9; that \( L_i = L(a_i^{(j-1)}, d_i) = L(A^{(i)}) \), as follows from Lemma 8; that if \( A = A^{(k)} \cdots A^{(i)} \) then \( L = L(A) \); and that the matrices \( A^{(i)} \) are \( d_i \)-simple and the lattices \( L_i \) are cyclic.

7. THE COMPLEXITY OF ALGORITHM SF

The number of iterations of the algorithm is bounded by \( \min(\log d, n) \), as mentioned in the proof of its correctness.

Steps 2 and 3 are \( O(\log^2 d) \), as determined by the \text{gud} operation.

Step 4 is \( O(n \log d) \), since modular division is equivalent to the \text{gcd} operation.

Step 5 is \( O(n) \).

It follows that the complexity of the algorithm is \( O(\log d (\log d + n) \min(\log d, n)) \). The size of the intermediate results is bounded by \( d^2 \), as is easy to determine by considering the various steps of the algorithm.

Remark 1. Given a lattice in the form \( L(B) \), one can apply to the matrix \( B \) the factorization algorithm \text{CF}, resulting in a factorization of \( L(B) \) into cyclic factors, and subsequently, by using Algorithm SF, one can factor every cyclic factor into simple factors, rendering a factorization of \( L(B) \) into simple factors \( L(A_j) \).
The total number of iterations of both algorithms, when applied in sequence, is still bounded by \( \log d \), where \( d \) is the absolute value of \( |B| \). This is implied by the fact that each time a new factor is produced, by either algorithm, the parameter \( d \) for the next iteration in Algorithm CF and the parameter \( d^{(j)} \) for the next iteration in the SF algorithm are reduced by a factor of at least 2.

REMARK 2. Given a factorization of a lattice \( L(B) \) into simple factors \( L(A_k) \cdots L(A_1) \), we can find a unimodular matrix \( U \) such that \( B = UA_k \cdots A_1 \). This follows from the fact that the rows of \( B \) and the rows of \( A = A_k \cdots A_1 \) span the same lattice.

8. A THIRD FACTORIZATION ALGORITHM

Algorithm CF receives at input a matrix \( B \) whose determinant in absolute value is equal to some integer \( d \). The factorization of \( d \) into prime factors is not required in the various steps of the algorithm. Similarly, factorization into prime factors is not required for the parameter \( d \), included in the input \( (a_1 \cdots a_n \ d) \), for Algorithm SF. The simple lattices output by Algorithm SF have the form \( L_j(a_1^{(j)}, \ldots, a_n^{(j)}, d_j) \), and for some \( i, \gcd(a_i^{(j)}, d_j) = 1 \).

If a factorization of \( d \) (and therefore also of \( d_j \), which divides \( d \)) into prime factors is given, then an additional factorization of the \( L_j \) lattices is possible.

The resulting factor lattices will be simple lattices such that the determinant of their bases, in absolute value, is equal to a prime number. The factorization is provided by Algorithm PF shown below. For the sake of simplicity, and without loss of generality, we assume that the lattice to be factored has the form \( L(a_1, \ldots, a_n, d) \) with \( \gcd(a_1, d) = 1 \), i.e., we assume that \( i = 1 \).

ALGORITHM PF. Input: \((a_1^{(0)} \cdots a_n^{(0)} d^{(0)}) := (a_1 \cdots a_n \ d)\) such that \( 0 \leq a_l < d \) for \( 1 \leq l \leq n \), and that \( a_1 \) is positive and satisfies \( \gcd(a_1, d) = 1 \); a factorization of \( d \) into prime and not necessarily distinct factors \( d = d_1 \cdots d_k \) is provided.

1. Set \( j = 1 \);
2. Define the simple matrix \( A^{(j)} = [s_{ij}^{(j)}] \) as the unit matrix except for its first column, whose first entry is equal to \( d_j \) and whose other entries are \( s_{lj}^{(j)} = -(a_l^{(j-1)} / a_1^{(j-1)}) \mod d_j \); output \( A^{(j)} \);
3. Reset
   \[
   a_l^{(j)} := \frac{1}{d_j} \left( s_l^{(j)} \cdot a_l^{(j-1)} + a_l^{(j-1)} \right), \quad 2 \leq l \leq n,
   \]
\[ a_1^{(j)} = a_1^{(j-1)}, \]
\[ d^{(j)} := d^{(j-1)}/d_j; \]
\[ \text{output } a^{(j)} = (a_1^{(j)} \cdots a_n^{(j)}) \left\{ a^{(j)} = (1/d_j)A^j a^{(j-1)} \right\}; \]

4. If \( d^{(j)} > 1 \) then set \( j := j + 1 \) and go to 2;

5. Halt.

Clearly \( d^{(j)} \) divides \( d^{(0)} \) and \( a_1^{(j)} = a_1^{(0)} \) all through the algorithm, implying

\[ \gcd(a_1^{(j)}, d^{(j)}) = 1 \quad \text{for} \quad 0 \leq j \leq k. \]

One can also prove that Lemmas 8 and 9 hold true for Algorithm PF in the same way as those lemmas have been proven to hold for Algorithm SF. The details are omitted.

**REMARK.** Given a lattice \( L(B) \). Assume that \( d \) is the absolute value of the determinant of \( B \), and assume that a factorization of \( d \) into prime factors (not necessarily distinct) is given, \( d = d_1 \cdots d_k \). By applying Algorithms CF, SF, and PF in sequence, one can factor \( L(B) \) into a product of lattices \( L(A_j) \) such that the absolute value of the determinant of \( A_j \) is \( d_j \), \( 1 \leq j \leq k \), and the matrices \( A_j \) are simple. The total number of iterations, for the three algorithms applied in sequence, is still bounded by \( \log^* d \), as is easy to see.

The factorization into "prime determinant" factors is not unique. In particular, the factorization depends on the order of the prime factor of the \( d \)-parameter input to Algorithm PF, when applied to the factors provided by Algorithm SF.

### 9. MODULAR LATTICES

An interesting application of the factorization algorithm SF described in Sections 5–7 will now be shown. We shall use the technique developed for that algorithm in order to find a modular basis for a modular lattice (see Definition 2 in Section 1).

**Lemma 10.** The number of points in \( L_d(a_1, \ldots, a_n) \) is equal to \( d^{n-1} \) for any modular lattice \( L_d(a_1, \ldots, a_n) \).

**Proof.** We prove by induction that the number of solutions over the integers of the equation \( \sum a_i x_i \equiv r \pmod{d} \), such that \( 0 \leq x_i < d \), is \( d^{n-1} \), given that \( \gcd(a_1, \ldots, a_n, d) = 1 \) and \( r \) is any integer.

**Basis.** If \( n = 1 \), then \( \gcd(a_1, d) = 1 \) implies that \( a_1 x \equiv r \pmod{d} \) has a unique solution, as is well known, in the range \( 0 \leq x < d \).
Step. We consider two cases:

(1) \( \gcd(a_2, \ldots, a_n, d) = 1 \). For any value \( c \) of \( x_1 \) the equation \( \sum_{i=2}^{n} a_i x_i \equiv -a_1 c + r \pmod{d} \) has, by induction, \( d^{n-2} \) solutions, and \( c \) can assume \( d \) values \( 0 \leq c < d \), resulting in \( d^{n-1} \) different solutions of \( \sum_{i=1}^{n} a_i x_i \equiv r \pmod{d} \), as required.

(2) \( \gcd(a_2, \ldots, a_n, d) = g > 1 \) and \( \gcd(a_1, g) = 1 \). Consider the equation \( \sum_{i=2}^{n} a_i x_i \equiv -a_1 x_1 + r \pmod{d} \). If \( x_1 \cdots x_n \) is a solution of this equation, then \( g \) must divide \(-a_1 x_1 + r\), since \( g \) divides the left-hand side and \( g \) divides \( d \). Thus \(-a_1 x_1 + r \equiv 0 \pmod{g} \) or \( x_1 \equiv r/a_1 \pmod{g} \) [since \( \gcd(a_1, g) = 1 \)]. This equation for \( x_1 \) has a unique solution in the range \( 0 \leq x_1 < g \) and has \( d' = d/g \) solutions in the range \( 0 \leq x_1 < d \). Let \( x_1 = c \) be such a solution, \( 0 \leq c < d \). Then \(-a_1 c + r = g s \), and the original equation, for this particular value of \( x_1 \) takes the form \( g \sum_{i=2}^{n} a_i' x_i \equiv gs \pmod{d'} \), where \( g a_i' = a_i, 2 \leq i \leq n \), and \( gd' = d \) (\( g \) divides \( a_i \), \( i \geq 2 \), and \( g \) divides \( d \)). This equation reduces to the equation \( \sum_{i=2}^{n} a_i' x_i \equiv s \pmod{d'} \). By the induction hypothesis, the reduced equation has \( (d')^{n-2} \) solutions in the range \( 0 \leq x_1 \leq d' \) [\( \gcd(a', d') = 1 \)]. Let \( (m_2 \cdots m_n) \) be such a solution, \( 0 \leq m_i \leq d' \). Then the vector \( (m_2 \cdots m_n) + d' (k_2 \cdots k_n) \), where the \( k_i \)'s are any integers in the range \( 0 \leq k_i < g \), solves the reduced equation, and therefore solves the original equation, and its entries are in the range \( 0 \leq m_i + d' k_i < d \). This brings the total number of solutions to \( (d')^{n-2} g^{n-1} = d^{n-2} g \) for the given value of \( x_1 \). But \( x_1 \) can assume \( d' \) different values. The total number of solutions is therefore \( d^{n-2} g d' = d^{n-1} \), as required.

We will show now how to find an \( n \times n \) matrix \( B \) whose rows are in \( L_d(a_1, \ldots, a_n) \) and whose determinant is equal to \( d \). The rows of such a matrix are a modular basis for \( L_d \) in the following sense: The volume of the parallelopiped spanned by the rows of \( B \) is equal to \( |B| = d \), and therefore exactly \( d^{n-1} \) translates of this parallelopiped can be packed, modulo \( d \), in the \( n \)-dimensional cube \( 0 \leq x_i \leq d, 0 \leq i \leq n \). We can thus represent the \( d^{n-1} \) elements of \( L_d(a) \) as modulo-\( d \)-linear combinations of the rows of \( B \), and the coefficients of the combinations \( c_i \) are in the range \( 0 \leq c_i < d \). We must consider all the cases listed below.

Degenerate cases:

Case 1.1. All the entries in \( a \) are zero except one. Assume \( a = (a_1, 0 \cdots 0) \), \( \gcd(a_1, d) = 1 \). Any point in \( L_d(a) \) must have its first coordinate equal to 0 in order to satisfy the modular equation [follows from the fact that \( \gcd(a_1, d) = 1 \)]. Any matrix \( B \) whose rows are in \( L_d(a) \) must have, therefore, its first column equal to the zero column, and its determinant cannot be equal to \( d \).
Case 1.2. Only two entries in $a$ are nonzero: assume $a_1$ and $a_2$, and $a_2 \equiv -a_1 \pmod{d}$. Any point in $L_d$ must satisfy the condition that $x_1 = x_2$ in order to satisfy the modular equation. Any matrix $B$ whose rows are in $L_d(a)$ has therefore two equal columns, the first and the second, and its determinant cannot be equal to $d$.

The above two cases are the only degenerate cases. We will show now that for any other $a$ we can find a matrix $B$ as required.

Regular cases:

Case 2.1. For some $a_i$, $\gcd(a_i, d) = 1$, $a$ has at least two nonzero entries.

Case 2.1.1. $a$ has exactly two nonzero entries: assume $a_1$ and $a_2$, and $a_2 \not\equiv -a_1 \pmod{d}$. Assume $\gcd(a_1, d) = 1$. Set $m \equiv -a_2/a_1 \pmod{d}$. Then $1 < m < d$ [since $a_2 \not\equiv -a_1 \pmod{d}$]. Set $k = \lfloor d/m \rfloor$. Then $1 < k < d$ (since $a_2 \not\equiv -a_1$, which implies that $d > 2$, and $1 < m < d$). The required matrix $B$ can be defined now as

$$B = \begin{bmatrix} m & 1 \\ km - d & k \\ & \ddots \end{bmatrix}.$$ 

Clearly $0 \leq km - d < d$, $|B| = d$, and $Ba^T \equiv 0 \pmod{d}$. $[0$ denotes here the 0-vector, $a = (a_1 a_2 0 \cdots 0)$, and the congruence is satisfied for every row of $B$].

Case 2.1.2. $a$ has three or more nonzero entries and $d = 2$. Assume $a_1 = a_2 = a_3 = 1$, $0 < a_i < 2$ for $4 \leq i \leq n$. The required matrix $B$ can be defined as

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ \vdots & \vdots & \vdots \\ a_n & & 1 \end{bmatrix}.$$ 

As is easy to verify $|B| = 2$ and $Ba^T \equiv 0 \pmod{2}$.

Case 2.1.3. $a$ has three or more nonzero entries and $d > 2$. Assume $a_1, a_2, a_3 > 0$, and assume that $\gcd(a_1, d) = 1$. Either $a_2 \not\equiv -a_1 \pmod{d}$ or $a_3 \not\equiv -a_1 \pmod{d}$ or $a_3 \not\equiv -a_2 \pmod{d}$, since if the first two alternatives do not hold, then the third alternative must hold given that $d > 2$. If the first two alternatives do not hold and
the third does, then from \( a_2 \equiv -a_1 \pmod{d} \) and \( \gcd(a_1, d) = 1 \) we get that \( \gcd(a_2, d) = 1 \). Up to permutation of indices we may assume therefore that \( a_2 \not\equiv -a_1 \pmod{d} \) and \( \gcd(a_1, d) = 1 \). Set \( m_i \equiv -a_i/a_1 \) for \( i \leq 2 \), and set \( k = \lfloor d/m_2 \rfloor \). As in case 2.1.1, we have that \( 1 < k < d \) and we can define the matrix \( B \) as

\[
B = \begin{bmatrix}
m_2 & 1 \\
km_2 - d & k \\
m_3 & 1 \\
\vdots & \ddots \\
m_n & 1
\end{bmatrix}.
\]

As in case 2.1.1, we have that that \( 0 \leq km_2 - d, m_2, m_3, \ldots, m_n < d, |B| = d, \) and \( Ba^T \equiv 0 \pmod{d} \), as required.

Case 2.2. For all \( a_i \neq 0 \), \( \gcd(a_i, d) > 1 \). We assume here that \( a \) has at least two nonzero entries (otherwise we have case 1.1). Using Algorithm SF we can find matrices \( A_1, \ldots, A_k \) (\( k \) is the number of iterations of the Algorithm SF) such that the matrix \( B = A_k \cdots A_1 \) has nonnegative entries, all the rows of \( B \) satisfy the modular equation \( \sum a_i x_i = 0 \pmod{d} \) (follows from Lemma 9), and \( |B| = d \) (as shown in the proof of correctness of Algorithm SF). We will show now that all the entries of \( B \) are less than \( d \), thus showing that \( B \), as defined above, satisfies all the required conditions. Let \( B = A_k \cdots A_1 \), and denote by \( B_j, 1 \leq j \leq k \), the matrix \( B_j = A_j \cdots A_1 \) (\( B_1 = A_1 \) and \( B_k = B \)). Recall that the \( A_i \) matrices are simple, i.e., every \( A_i \) is a unit matrix except for one column. At every iteration of Algorithm SF the running index \( i \) (see step 5) is increased, so that the location of the nonunit column in \( A_i \) differs from the location of that column in \( A_1 \) when \( i \neq 1 \). W.l.o.g. we shall assume that the nonunit column of \( A_1 \) is the \( i \)th column. We shall also assume that \( k \geq 2 \), which is implied by the assumption that \( a \) has at least two nonzero entries and by the assumption that \( \gcd(a_i, d) > 1 \) for all nonzero \( a_i \). Let \( D_j \) denote the number \( D_j = d_j \cdots d_1 \) (\( D_1 = d_1, D_k = d \)).

**Lemma 11.** All the entries of \( B_j \) are less than \( D_j \) for \( 2 \leq j \leq k \).

**Proof.** By induction

**Basis** \( j = 2; \)

\[
B_2 = A_2 A_1 = \begin{bmatrix}
1 & s_{1, 2}^{(2)} \\
d_2 & s_{2, 2}^{(2)} \\
\vdots & \ddots \\
s_{n, 2}^{(2)} & 1 \\
s_{1, 2}^{(2)} & \cdots \\
s_{2, 1}^{(1)} & 1 \\
s_{3, 2}^{(2)} & \cdots \\
s_{n, 1}^{(1)} & 1
\end{bmatrix}
\]
The second row of $B_2$ is equal to $d_2(s_{21}^{(1)}, 1, 0, \ldots, 0)$. The two nonzero entries in this row are $d_2s_{21}^{(1)}$ and $d_2$, and they are both less than $D_2 = d_2d_1$, since $s_{21}^{(1)} < d_1$ by construction, and $d_1 \geq 2$. The other rows of $B_2$, except the first, have the form

$$\begin{pmatrix} s_{11}^{(1)} & 0 & \cdots & 1 & 0 & \cdots & 0 \end{pmatrix} + s_{12}^{(2)} \begin{pmatrix} s_{21}^{(1)} & 1 & 0 & \cdots & 0 \end{pmatrix}, \quad l \neq 2.$$  

The first entry in such a row is $s_{11}^{(1)} + s_{12}^{(2)}s_{21}^{(1)} \leq d_1 + (d_2 - 1)(d_1 - 1) = d_2d_1 - d_2 + 1 < d_2d_1 = D_2$ (since $d_2 \geq 2$). The second entry is $s_{12} < d_2 < D_2$, and the $l$th entry is $1 < D_2$. A similar argument works for the first row.

**Step.** Assume now that all the entries in $B_{j-1}$ are less than $D_{j-1}$, $j - 1 \geq 2$. Then

$$B_j = \begin{bmatrix} 1 & s_{1j}^{(j)} & \cdots & 1 \\ \vdots & s_{j-1,j}^{(j)} & \ddots & \vdots \\ 1 & s_{j,j}^{(j)} & \ddots & 1 \\ s_{j+1,j}^{(j)} & \vdots & \ddots & 1 \\ \vdots & \vdots & \ddots & 1 \\ s_{nj}^{(j)} & \cdots & 1 \end{bmatrix} B_{j-1}.$$  

Let $B_l = (b_{l1} \cdots b_{ln})$ denote a typical row of $B_{j-1}$. By assumption $b_{li} \leq D_{j-1} - 1$ for $0 \leq i \leq n$. The $j$th row of $B_j$ is $d_j(b_{j1} \cdots b_{jn})$ with $d_jb_{ji} < d_jD_{j-1} = D_j$, $1 \leq l \leq n$.

The $l$th row of $B_j$, with $l \neq j$, has the form

$$s_{lj}^{(j)}(b_{jl} \cdots b_{jn}) + (b_{l1} \cdots b_{ln})$$

Now $s_{lj}^{(j)}b_{jn} + b_{lm} \leq (d_{j-1})(D_{j-1} - 1) + D_{j-1} - 1 = d_j(D_{j-1} - 1) < d_jD_{j-1} = D_j$. The proof is now complete. 

Since all the possible cases have been considered, we can sum up the previous results in the theorem below.

An $n$-dimensional vector with nonnegative entries $a = (a_i)$ is called **degenerate** with regard to an integer $d$ if $a$ is the zero vector, or it has a single nonzero entry $a_i < d$ or it has exactly two nonzero entries $a_i$ and $a_j$ such that $a_i = -a_j \pmod{d}$.

**Theorem.** Given a modular lattice $L_d(a)$ such that $a$ is not degenerate with regard to $d$, a basis $B$ for $L_d(a)$ can be found in polynomial time.
10. APPENDIX. A BLANKINSHIP-TYPE ALGORITHM [1]

In this appendix we present an algorithm for the following problem: Given a vector of integers \(a = (a_1 \cdots a_n)\) such that \(\gcd(a_1, \ldots, a_n) = 1\), find a unimodular matrix \(A\) such that \(AA^T = (0 \cdots 0 1)^T\). Let \(M\) be the maximal entry in \(a\). The algorithm has the following properties: All the entries in the first \(n - 1\) columns of \(A\) are bounded by \(M\), the intermediary results are bounded by \(nM^n\), the complexity of the algorithm is \(O(n^3 + \log M)\).

**ALGORITHM BL**

1. Given \(a = (a_i)\), construct the \(n \times (n + 1)\) matrix \(B = [b_{ij}]\) whose first column is \(a^T\) and last \(n\) columns are the unit matrix \(I\).

2. First iteration.
   2.1. Find two integers \(u_1, v_1\) such that \(|u_1| < a_n, |v_1| < a_1\), and \(u_1 a_1 + v_1 a_n = \gcd(a_1, a_n) = g_2\). Denote by \(b_j\) the rows of \(B\).
   2.2. Reset \(b_n := u_1 b_1 + v_1 b_n; b_1 := a_n b_1 - a_1 b_n\). (Since initially \(b_{11} = a_1\) and \(b_{1n} = a_n\), we have, after the reset, that \(b_{11} = 0\) and \(b_{1n} = g_2\). The transformation is a unimodular transformation, since it can be described as \(B := UB\), where \(U\) is a matrix whose first row is \((u_1 0 \cdots 0 v_1)\), whose last row is \((a_n 0 \cdots 0 -a_1)\), and all whose other rows are unit vectors: for \(i \neq 1, n\), the \(i\)th row has its \(i\)th coordinate equal to one and all other coordinates zero. \(|U| = 1\), as follows from its definition.)

   \{The first and last row of \(B\) before step 2.1 had the form
   \[b_1 = (a_1 1 0 \cdots 0), \quad b_n = (a_n 0 \cdots 0 1)\].

   After step 2.1 the same rows are changed into
   \[b_1 = (0 a_n 0 \cdots 0 -a_1), \quad b_n = (g_2 u_1 0 \cdots 0 v_1)\]

   with \(u_1 < a_n\). We have also that \(b_2 = (a_2 0 1 0 \cdots 0)\).

3. Iteration \(i, 2 \leq i \leq n - 1\). Assume that before the \(i\)th iteration, \(2 \leq i \leq n - 1\), the \(i\)th and the last row of \(B\) have the form

   \[b_i = (a_i 0 \cdots 0 1 0 \cdots 0), \quad b_n = (g_i b_{n, i} 0 \cdots 0 b_{n, n+1})\]

   and

   \[b_{nj} < b_{jj} \text{ for } 2 \leq j \leq i, \quad g_i = \gcd(a_1, \ldots, a_{i-1}, a_n)\].

   \{This condition holds for \(i = 2\).}
3.1(i) Find two integers \( u_i, v_i \) such that \( |u_i| < g_i, \) \( |v_i| < a_i \), and \( u_i a_i + v_i g_i = \gcd(a_i, g_i) = \gcd(a_1, \ldots, a_i, a_n) = g_{i+1} \).

3.2(i) Reset

\[
\begin{align*}
b_n &:= u_i b_i + v_i b_n, \\
b_i &:= g_i b_i - a_i b_n
\end{align*}
\]

{After the reset we have that \( b_{i+1} = 0 \) and \( b_{n+1} = g_{i+1} \) as explained in step 2.2. The transformation is unimodular.}

3.3(i) For \( j := i \) down to 2 reset

\[
\begin{align*}
b_i &:= b_i - \frac{b_{ij}}{b_{j-1,j}} b_{j-1}, \\
b_n &:= b_n - \frac{b_{nj}}{b_{j-1,j}} b_{j-1}
\end{align*}
\]

4. Output the matrix consisting of the last \( n \) columns of \( B \), to be denoted by \( A \).

End of algorithm.

**Correctness**

If \( B_0 \) is the \( B \) matrix at input and \( A' \) is the matrix representing the composition of all unimodular transformations performed during the algorithm, then

\[
B_0 = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \quad \text{and} \quad A' B_0 = \begin{bmatrix} 0 \\ \vdots \\ A \\ 0 \end{bmatrix},
\]

as follows from the definitions and from the steps of the algorithm. Therefore,

\[
A' = A \quad \text{and} \quad A \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix},
\]

as required.

**Complexity and Size of Intermediate Results**

Let \( M \) be the maximal integer among \( a_1, \ldots, a_n \). As shown in Bradley [2], steps 2.1 and 3.1(i), \( 2 \leq i \leq n - 1 \), are \( O(\log M + n) \) altogether.

The \( O(n) \) iterations in steps 2.2, 3.2 (i) are \( O(n) \) each, contributing \( O(n^2) \) operations.

The \( O(n) \) iterations in steps 3.3(i) are \( O(n^2) \), contributing \( O(n^3) \) operations.
The total number of operations is therefore $O(\log M + n^3)$.

In the application of this algorithm to the problem considered in the paper we shall need only columns 1 to $n - 1$ of the matrix $A$ output at step 4. We shall ignore therefore the last column of $A$ (and of $B$) in the discussion below. In the algorithm itself the computation of the values of the entries in the last column of $B$ can be ignored, and those values, if needed, can be computed at the very end: based on the requirement that $Aa^T = (0 \cdots 0 1)^T$, the values of the last column of $A$ can be found easily if all the other entries of $A$ are given.

Assume that before iteration $i$ all the entries in the first column of $B$ are bounded in absolute value by $M$ (the maximal coordinate of $a$). We will show that this property is restored when the $i$th iteration is completed while the intermediate values of those entries during the iteration are bounded by $nM^n$.

The rows affected by the $i$th iteration are the $i$th row and $n$th row. Before the iteration those rows have the form

$$
\begin{align*}
b_i &= (a_i 0 \cdots 0 1 0 \cdots 0), \\
n_b &= (g_i b_{n,2} \cdots b_{n,i} 0 \cdots 0 b_{n,n+1}),
\end{align*}
$$

where $a_i$, $g_i$, $b_{n,j}$ for $2 \leq j \leq i$ are all bounded by $M$ [recall that $g_i = \gcd(a_1 \cdots a_{i-1}, a_i)$] after step 3.2; $b_i$ and $b_n$ change to

$$
\begin{align*}
b_i &= (0 -a_i b_{n,2} \cdots -a_i b_{n,i} g_i 0 0 \cdots 0 -a_i b_{n,n+1}), \\
n_b &= (g_{i+1} v_i b_{n,2} \cdots v_i b_{n,1} u_i 0 0 \cdots 0 v_i b_{n,n+1}).
\end{align*}
$$

Now $|u_i| < g_i \leq M$ and, since $|v_i| < |a_i|$, all the entries in the new $b_i$ and $b_n$ (except the last) are bounded by $M^2$.

Consider now step 3.3(i) for $j = i$. Then $b_{i-1}$ has the form $b_{i-1} = (0 b_{i-1,2} \cdots b_{i-1,i} 0 \cdots 0 b_{i-1,n+1})$ with $b_{i-1,k} \leq M$ for all $k$. Thus, $|b_{ii}/b_{i-1, i}| \leq M^2$, and therefore, after step 3.3(i) for $j = 1$, we have $b_{i,i} < b_{i-1,i} \leq M$ and $b_{i,k} \leq 2M^2$ for $k < i$. Similarly, $b_{n,i} < b_{i-1,i} \leq M$ and $b_{n,k} \leq 2M^2$ for $k < i$. Using a similar argument, we get after step 3.3(i) for $j = i - 1$ that $b_{i,i-1}, b_{n,i-1} < b_{i-2,i-1} \leq M$ and $b_{i,k}, b_{n,k} < 3M^3$ for $k < i - 1$, and similarly for $j = i - 2$ etc.

The loop 3.3(i) will therefore keep the intermediate results bounded by $nM^n$, and when the loop is completed all the entries in $B$ (except the last column) are bounded by $M$.

**Corollary.** The entries in the first $n - 1$ columns of the matrix $A$ output by the algorithm are bounded by $M$. Moreover, if the last column and the last row of $A$ are removed, then the remaining matrix is lower triangular and the entries in any column of the resulting matrix are nonnegative and bounded by the diagonal entry in that column.
REFERENCES


Received 24 November 1991; final manuscript accepted 26 March 1993